Universidade do Minho
Escola de Engenharia

Paula Correia Tavares

O Impacto da Animação e da Avaliação Automática na Motivação para o Ensino da Programação

outubro de 2017

Paula Correia Tavares **O Impacto da Animação e da Avaliação Automática na Motivação para o Ensino da Programação** UMinho | 2017

Universidade do Minho
Escola de Engenharia

Paula Correia Tavares

O Impacto da Animação e da Avaliação Automática na Motivação para o Ensino da Programação

Tese de Doutoramento em Informática

Trabalho efetuado sob a orientação do
Pedro Rangel Henriques
(Professor Associado com Agregação)
e da
Elsa Maria Ferreira Gomes
(Professora Adjunta)

outubro de 2017

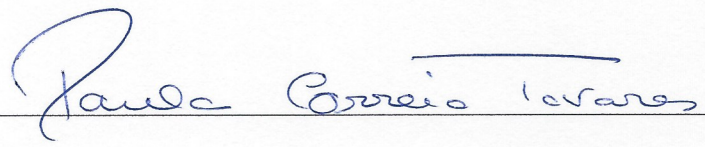
DECLARAÇÃO DE INTEGRIDADE

Declaro ter atuado com integridade na elaboração da presente tese. Confirmo que em todo o trabalho conducente à sua elaboração não recorri à prática de plágio ou a qualquer forma de falsificação de resultados.

Mais declaro que tomei conhecimento integral do Código de Conduta Ética da Universidade do Minho.

Universidade do Minho, 27 de outubro de 2017

Nome completo: Paula Correia Tavares

Assinatura: 

DECLARAÇÃO DE INTEGRIDADE

Declaro ter atuado com integridade na elaboração da presente tese. Confirmo que em todo o trabalho conducente à sua elaboração não recorri à prática de plágio ou a qualquer forma de falsificação de resultados.

Mais declaro que tomei conhecimento integral do Código de Conduta Ética da Universidade do Minho.

Universidade do Minho, 27 de outubro de 2017

Nome completo: Paula Correia Tavares

Assinatura:

“Ninguém ensina nada a ninguém, no máximo ajuda-se o outro a aprender”

Pensamentos de Galileu Galilei

Agradecimentos

Gostaria de agradecer a todos os que, de uma forma direta ou indireta, contribuiram para que fosse possível atingir o final desta minha caminhada.

Em primeiro lugar não posso deixar de prestar o meu profundo agradecimento ao Professor Pedro Rangel Henriques por toda a dedicação, tempo disponibilizado, persistência e entusiasmo. Sinto-me verdadeiramente previligiada e afortunada por o ter como orientador. Foi um caminho duro mas sem a sua ajuda teria sido seguramente muito mais difícil. Obrigada!

À Professora Elsa Gomes, o meu obrigada por todo o suporte e encorajamento, sempre presente e diligente nas suas observações tão pertinentes e oportunas. Foi sem dúvida um pilar importante nesta caminhada estreitando laços de amizade.

À Universidade do Minho, em particular ao PDINF, por me ter concedido a oportunidade da realização deste trabalho.

Ao ISEP e em particular ao Departamento de Engenharia Informática, pelas facilidades concebidas que possibilitaram a conclusão desta dissertação.

Ao Luís Barreiro e Pedro Duarte pelo trabalho efetuado na implementação do PEP.

À Bertil e ao Marílio o meu obrigada pela amizade demonstrada ao longo destes anos e pela paciência e apoio incondicional.

À Rosa Reis pela ajuda dada, principalmente na difícil fase final.

Aos colegas com quem trabalho, em particular os das unidades curriculares que leciono, por todo o apoio e compreensão ao longo deste percurso.

Ao Miguel, pelo grande “empurrão” que me permitiu chegar até aqui!

Às pessoas cujo convívio é muito anterior ao início desta caminhada e que foram importantes para aqui chegar. Em particular à minha família, porto de abrigo, nos bons, mas principalmente nos momentos mais difíceis, por todo apoio, compreensão e incentivo.

Ao meu pai e à minha mãe pelo apoio total e incondicional. Sem a sua ajuda de certeza que tudo seria mais difícil. Obrigada! Foi o seu exemplo inspirador que me permitiu aqui chegar. Sem eles não teria conseguido.

À João, um exemplo de força, pelas palavras encorajadoras! Sempre.

Obrigad Fernando. Mesmo com o seu silêncio, o seu exemplo de persistência permitiu-me perceber que somos sempre capazes. O caminho foi duro e longo mas a compreensão esteve sempre presente.

À Rita, pelas muitas horas que lhe "roubei" e pela sua compreensão. Que este seja um esforço inspirador para a sua realização pessoal e profissional.

Resumo

A aprendizagem da programação é uma tarefa complexa que coloca desafios importantes. Nesta dissertação, propõem-se duas abordagens ao processo de ensino nas disciplinas de Programação que visam aumentar a motivação dos alunos e a sua auto-estima.

As propostas apresentadas resultam de um estudo cuidado das dificuldades que os alunos sentem ao contactar com a programação de computadores, quer a nível do raciocínio lógico que é necessário para esquematizar a resolução de um qualquer problema, quer a nível da compreensão da linguagem de programação que terá de ser usada para exprimir essa resolução. Percebendo-se que a principal razão assenta na falta de motivação que resulta em parte da falta de confiança e quebra da autoestima necessárias para ultrapassar os impasses, procedeu-se também ao estudo dessa área da psicologia, conforme aqui se relata. Identificados os problemas inerentes à motivação humana e às caraterísticas do processo de resolução de problemas por computador, procuraram-se técnicas que tenham vindo a ser propostas para ajudar os alunos na aquisição de conhecimentos de Programação. Conforme se verifica ao longo do documento focou-se a atenção em duas estratégias: a capacidade do sistema visual humano para rapidamente apreender conceitos e sobretudo processos; a necessidade de os alunos receberem rápido *feedback* quando se aventuram a resolver sozinhos um problema. No primeiro caso investigaram-se os sistemas de Animação de Programas e no segundo caso os sistemas de Auto-avaliação de Programas. As duas abordagens propostas baseiam-se precisamente numa combinação destas duas estratégias. Dessas abordagens, uma foi alvo de experimentação em sala de aula para se poder aferir o seu real impacto. A outra serviu de base para a proposta de uma plataforma Web para suporte ao processo de ensino/aprendizagem da Programação; o sistema designado por PEP chegou a ser prototipado, conforme é descrito. Para se poder avaliar a qualidade do PEP e até guiar a implementação de uma versão final foram estudados sistemas de aferição de qualidade de *software* para o ensino, com especial destaque para a *framework* QEF.

Por fim ainda se estudou a importância de incluir no sistema de apoio ao ensino elementos dos jogos que estimulam a motivação e ainda se propôs um enriquecimento do sistema PEP através da inclusão de técnicas de ludificação ou gamificação.

Palavras-Chave

Programação; Aprendizagem; Motivação e Auto-confiança dos alunos; Animação de Programas; Avaliação Automática de Programas; *Feedback* imediato; Gamificação.

Abstract

Learning programming is a complex task that raises important challenges. In this dissertation, two approaches for teaching Programming courses are proposed aimed at increasing students' motivation and their self-confidence/self-regulation .

The proposals presented came out after a deep study of the difficulties that students feel when they are beginning a computer programming course, either at the level of the logical reasoning that is necessary to sketch the resolution of a problem, or at the level of understanding the programming language that has to be used to code this resolution. Realizing that the main reason is based on the lack of motivation derived from the lack of confidence and self-regulation necessary to overcome the troubles, the area of psychology that studies motivation was also researched, as will be discussed in this dissertation. After identifying the problems inherent to human motivation and the characteristics of the computer problem solving process, techniques that have been proposed to help students on learning programming were surveyed. This study was focussed on two strategies: the power of human's to quickly and easily grasp concepts from static or dynamic visualizations; the positive impact of feedback returned to the students when they write a program on their own. In the first case, Program Animation systems were investigated; and in the second case, Automatic Program Evaluators were studied. The two approaches that outcame from this Ph.D. work are precisely based on a combination of these two strategies. Experiments in the classroom were drawn and conducted to validate the first approach. The second approach inspired the design of a Web-based platform (PEP) to support the teaching / learning process of Programming. A prototype of PEP, based on those guidelines and requirements, was developed by a team of M.Sc. Students as will be described. In order to evaluate PEP's quality, and to guide its final implementation, *software* quality assessment systems were studied, with special emphasis on QEF *framework*.

The importance of including, in the educational support systems, elements traditionally used in the context of games to stimulate the motivation was studied and an enrichment of PEP tool through the inclusion of techniques of *gamification* was still proposed.

Keywords

Programming; Learning; Students' motivation and Self-confidence; Program Animation; Automatic Program Evaluation; Immediate Feedback; Gamification.

ÍNDICE

ÍNDICE DE FIGURAS

ÍNDICES DE TABELAS

1. INTRODUÇÃO

A aprendizagem da programação é uma tarefa complexa que coloca desafios importantes aos docentes de informática. Muitos alunos confundem aprender a programar com aprender a sintaxe de uma qualquer linguagem de programação. Programar é, antes de mais, delinear estratégias que permitam resolver problemas, independentemente da linguagem usada. Porém o paradigma de programação adotado e a linguagem usada têm impacto, quer no processo de aprendizagem, quer no desempenho dessa tarefa (ACM/IEEE, 2013).

Neste trabalho de doutoramento, o problema que se pretende estudar é precisamente *o ensino da programação*.

1.1 Enquadramento e Motivação

A Programação tem por fim resolver problemas por computador, ou seja, visa preparar o computador para transformar dados em resultados que correspondam à solução do problema que se quer resolver; esses resultados são as respostas às questões colocadas pelo utilizador. Neste processo, o Programador desenvolve o programa que instrói o computador, o Utilizador final fornece os dados correspondentes à questão a resolver, e o Computador executa rápida e sistematicamente a transformação dos dados fornecidos em resultados de acordo com as regras estabelecidas pelo programa. A tarefa de Programação é muito complexa para

qualquer pessoa e por isso, para que seja bem-sucedida, costuma organizar-se em várias etapas:

- Análise do problema a resolver (normalmente enunciado de forma imprecisa e incompleta em língua natural);
- Criação de uma especificação, ou seja, de uma descrição rigorosa e não ambígua do problema, contendo todos os requisitos e restrições;
- Desenvolvimento do Algoritmo que que satisfaça todos os requisitos especificados;
- Codificação do Algoritmo usando uma Linguagem de Programação que o computador esteja habilitado a compreender;
- Execução e Testes ao Programa.

A utilização de linguagens de programação exige que se siga a sua sintaxe estritamente, pois o mais pequeno erro pode inviabilizar o funcionamento de um programa. O computador só percebe e executa as instruções que estejam escritas corretamente numa linguagem formal por ele entendida. Os algoritmos não são diretamente executáveis visto que a forma como são descritos é um pouco mais livre, mais abstracta que o nível executável pelo computador, ainda que deva ser suficientemente exata para que não haja qualquer ambiguidade.

As dificuldades encontradas pelos alunos quando iniciam a sua aprendizagem na programação mostram um vasto conjunto de carências a diferentes níveis, as quais não são, muitas vezes, resolvidas eficazmente pelos métodos de ensino e aprendizagem clássicos, resultando numa taxa elevada de insucesso (Hundhausen et al, 2000). As principais dificuldades desses alunos são (Proulx, 2000):

- Compreender o problema, devido à sua falta de familiaridade com o assunto ou devido à incapacidade para interpretar o enunciado do problema (identificar o seu significado);
- Pensar de uma maneira lógica para decompor o problema dado em partes cada vez menores e escrever o algoritmo correto (sequência de operações inequívocas e elementares) para resolvê-lo;
- Aprender a sintaxe da linguagem formal de programação e a sua semântica.

Aprender a programar torna-se assim uma tarefa bastante difícil para os alunos (conduzindo normalmente a um número grande de reprovações); do que foi dito acima percebe-se que uma das grandes dificuldades reside na compreensão e na aplicação de noções básicas para que o algoritmo resolva problemas concretos.

Como a experiência claramente demonstra, qualquer aluno só pode aprender a programar, programando, ou seja, só programando poderá aprender estratégias de resolução de problemas. Um comportamento ativo por parte do aluno, ao invés dum comportamento passivo, leva a uma melhoria da compreensão e resolução dos problemas propostos. Esta aprendizagem é considerada um processo iterativo, ou seja, programar aprende-se através da construção de nova informação sobre informação mais básica.

As dificuldades acima referidas, reconhecidas no processo de aprendizagem da programação, levaram à criação de linguagens e ao desenvolvimento de ambientes que facilitam a concepção de algoritmos, a escrita e análise dos programas. No entanto, tanto quanto sabemos, este problema ainda não está satisfatoriamente resolvido.

Novas abordagens ao ensino/aprendizagem devem ser tomadas em conta. O acesso à educação assistida por computador especialmente adaptada para atividades de programação deve ser explorado.

Por este motivo, vários autores (Hansen et al., 1999; Stasko et al., 1996; Hundhausen et al., 2002) dedicaram-se a estudar a eficácia pedagógica da animação e visualização de programas e ao desenvolvimento de algumas ferramentas, em que o potencial do sistema visual humano é o ponto de partida. Vários estudos foram feitos para verificar se, efetivamente, o uso da animação facilita a aprendizagem de algoritmos.

Muitas vezes os alunos entendem com alguma facilidade algoritmos previamente desenvolvidos, mas apresentam grandes dificuldades quando tentam desenvolver um programa de raiz. A animação de algoritmos pode ajudar os alunos a entender os programas, pode facilitar a análise de programas existentes e pode mesmo facilitar o desenvolvimento de novos.

Neste contexto, foram criados vários sistemas (Hundhausen et al., 2002) acreditando que as animações ajudavam na aprendizagem efetiva dos alunos e que forneciam facilidades aos

utilizadores de visualizar e interagir com a ferramenta de suporte. Porém, após vários experimentos e respectivas análises, tornou-se claro para o autor citado que o simples uso de animações de programas não consegue melhorar a aprendizagem como seria de esperar. Existem, portanto, nesta direção ainda uma série de questões por resolver e a considerar.

Pela sua experiência letiva, muitos professores reconhecem que, na maioria das vezes, os alunos, perante um problema, não entendem o que se pretende, não conseguem iniciar a resolução do problema e, quando não desistem, cometem erros que os desmotivam. Por isso e como já foi referido, uma das preocupações a ser abordada neste trabalho será averiguar quais as principais dificuldades intrínsecas à aprendizagem da programação.

Estas dificuldades na aprendizagem incentivaram a criação de ferramentas que permitem facilitar ao aluno a concretização de ideias mais abstratas e consequentemente a utilização de conceitos de programação.

A identificação e estudo de ferramentas, cujo interesse é considerado mais relevante para apoio ao ensino e aprendizagem da programação, será outro ponto a ser desenvolvido. Assim sendo, a animação e a visualização de programas que, tal como já abordado acima, ajuda a compreender a semântica da linguagem, ajuda no *debugging* e ajuda na compreensão dos algoritmos, será pois uma das vertentes a explorar e a estudar. Porém, será também importante perceber a razão do insucesso das primeiras tentativas do uso da visualização e da animação na aprendizagem da programação.

Além disso e tal como já foi afirmado, é muito importante dar aos alunos a oportunidade de praticar e resolver exercícios de programação por si mesmos desde o primeiro dia. Receber *feedback* é essencial para a aquisição de conhecimento (Verdú et al., 2011). O *feedback* imediato é importante para incentivar o aluno a progredir dando-lhe uma avaliação do trabalho em curso, indicando se o resultado que o seu programa está a produzir está certo ou errado e, se possível, explicando as causas do erro ou pelo menos descrevendo as situações em que o programa falha.

Novas ferramentas surgiram (nomeadamente na área de concursos de programação) para permitir a submissão de soluções (programas desenvolvidos pelos alunos) aos exercícios propostos pelo professor e proceder automaticamente à sua validação/avaliação, retornando

imediatamente informações sobre a solução apresentada (resposta submetida) (Verdú et al., 2011). Estas ferramentas podem ser incorporadas com sucesso nas atividades de ensino, permitindo que os alunos possam testar o seu trabalho recebendo de imediato informação relativa à correção do mesmo. Os Sistemas de avaliação automática, como são chamados, melhoram significativamente o desempenho dos alunos pois atuam de várias formas nos factores que estimulam a sua motivação ou auto-estima/auto-confiança. Esta abordagem aumenta o seu envolvimento e, consequentemente, contribui para melhorar o seu sucesso académico (Joy et al., 2005).

1.2 Objetivos

Pretende-se apresentar nesta secção, concisamente, os objetivos do trabalho de doutoramento que têm vindo a ser referidos ao longo do capítulo.

Com o fim de aumentar a motivação e autoconfiança dos alunos dos cursos introdutórios de programação, e consequentemente aumentar o sucesso deste processo, o objetivo deste trabalho de pesquisa é:

- identificar as dificuldades que realmente surgem neste processo de ensino/aprendizagem, estudando a motivação humana e os fatores intrínsecos e extrínsecos que a controlam;
- identificar e analisar com profundidade abordagens diferentes, apoiadas no uso do computador, que têm sido propostas para superar essas dificuldades;
- propor estratégias para melhorar o sucesso dos cursos de programação direcionadas para a motivação dos alunos. Estas estratégias serão baseadas no uso do computador e aplicações informáticas que possam aumentar o envolvimento dos alunos em tarefas de compreensão e desenvolvimento. Essas propostas, baseadas nos estudos teóricos fundamentados na literatura nos estudos práticos realizados, deverão combinar várias ferramentas originalmente criadas com objetivos diferentes;
- efetuar estudos experimentais com os alunos, para avaliar a eficácia das estratégias propostas acima.

1.3 Hipótese de Investigação

A hipótese de trabalho que se pretende provar ao longo desta dissertação é:

> Combinando a animação de programas e a avaliação automática de programas é possível aumentar a motivação, o envolvimento e o desempenho dos alunos de Programação conduzindo a um incremento no sucesso do processo ensino/aprendizagem das disciplinas nessa área.

1.4 Método de Investigação

Considerando o *sucesso* como a consequência final, a atingir, o *desempenho*, o *envolvimento* e a *motivação* serão os antecedentes a garantir.

Assim, para provarmos a hipótese de investigação formulada, entendemos que:

- O sucesso de um aluno consiste em obter aprovação à disciplina; aumentará o sucesso do processo de ensino-aprendizagem de uma disciplina de programação se o rácio '*aprovados/inscritos'* aumentar. Este rácio será a medida de avaliação do sucesso.
- O desempenho de um aluno consiste na sua capacidade de resolver corretamente o problema que lhe é colocado e no esforço gasto para o conseguir; aumentará o desempenho se aumentar o número de problemas que resolve corretamente e diminuir o esforço para tal. O número de respostas corretas a um conjunto de perguntas, num intervalo de tempo, será uma das edidas de avaliação do desempenho.
- O envolvimento de um aluno consiste no seu empenho em tentar resolver o problema que lhe é colocado; aumentará o envolvimento se continuar a tentar encontrar a solução correta após situações de fracasso não desistindo às primeiras dificuldades. Para aferir este ponto, deve-se contar o número de submissões que o aluno faz na ferramenta de Avaliação Automatica após o primeiro erro na busca de resolver bem o problema.
- A motivação de um aluno é um estado psicológico que o leva a envolver-se numa tarefa; se aumentar, consequentemente o envolvimento aumenta. A medida de avaliação identificada acima pode ser utilizada neste caso.

Note-se que zero (0) submissões após um retorno de mensagem de erro, é sinal claro de falta de envolvimento por falta de motivação ou auto-confiança.

Para além dos indicadores de envolvimento/motivação atrás mencionadas, considera-se que as plataformas de apoio fornecem outros dados que podem também ser analisados, como por exemplo: o numero de vezes que o aluno recorreu ao sistema de animação quando este era opcional; o número de exemplos ou exercícios que o aluno optou por ver/realizar quando estes eram opcionais, etc.

Sendo que o sucesso da disciplina não é mensurável no espaço físico-temporal de um doutoramento, a aferição dos restantes três critérios no contexto do trabalho de doutoramento será realizada através de experimentos conduzidos em sala de aula, medindo os registos (tais como o tempo gasto em cada tarefa) fornecidos pelas ferramentas de apoio que irão ser utilizadas e pela recolha de opiniões dos alunos através de inquéritos. É ainda típico na condução de experimentos considerar a opinião dos observadores (docentes neste caso) que fazem o acompanhamento.

1.5 Estrutura da Dissertação

Após se ter caracterizado o contexto em que se desenvolveu este trabalho de doutoramento, feita a sua motivação e definidos os objetivos, o Capítulo 2 analisa o processo do ensino/aprendizagem da Programação fazendo uma introdução à motivação do Ser Humano, à motivação dos alunos do Ensino Superior e à necessidade de lhes estimular a auto-estima ou auto-confiança; nesse capítulo e a propósito das técnicas a usar para incentivar a motivação, fala-se ainda sobre o uso da gamificação na educação e em como esta abordagem pode alterar o desempenho dos alunos. De forma a comprovar as principais dificuldades sentidas pelos aprendizes, identificadas ao longo do capítulo, é ainda apresentado um inquérito aplicado aos alunos de uma disciplina do primeiro ano de algoritmia e programação e feita a análise dos respostas recolhidas.

O Capítulo 3 descreve algumas ferramentas de Animação que poderão ser utilizadas para apoio aos alunos e professores, referenciando alguns autores que têm desenvolvido, ao longo de vários anos, estudos numa tentativa de solucionar e minorar as dificuldades dos alunos neste processo de ensino/aprendizagem apelando ao potencial do sistema visual humano. É

importante perceber, também, o impacto que, segundo alguns autores, estas ferramentas de animação tiveram efetivamente sobre a motivação e aprendizagem dos alunos, conforme se discute no Capítulo 4 .

No capítulo 5 abordam-se quatro ferramentas de Avaliação Automática de Programas, com o intuito de dar a possibilidade aos alunos de resolver exercícios por si mesmos obtendo então *feedback* imediato; em simultâneo estas ferramentas dão aos professores a capacidade de fazer uma avaliação mais rápida e eficaz, aumentando assim o tempo útil na sala de aula. Este estudo será a base de suporte da escolha de uma das ferramenta usada nos experimentos, a serem apresentados no Capítulo 7.

No Capítulo 6 são expostas duas abordagens ao ensino da programação, suportadas em recursos informáticos (Animação de Programas e Avaliação Automática de Programas), com o objetivo de aumentar a motivação e a autoconfiança dos alunos destes curso. Apresentam-se as diferentes etapas que compõem as abordagens, descrevendo em pormenor cada uma delas. Para cada uma das abordagens foram feitos estudos experimentais, no âmbito escolar, que se descrevem e analisam no já referido Capítulo 7, para validação dessas mesmas propostas e para melhor perceber como diminuir as reais dificuldades dos alunos nessa fase de aprendizagem.

Seguindo uma das abordagens apresentada no Capítulo 6, foi desenvolvido uma ferramenta informática (PEP) que permite ao aluno fazer as suas sessões de auto-estudo fora das aulas ou que permite apoiar o professor nas suas aulas, fornecendo *feedback* sobre a correção de cada programa desenvolvido e coletando informação sobre como decorreu cada sessão de trabalho de cada aluno. A apresentação dessa plataforma PEP é o assunto do Capítulo 8.

No Capítulo 9 é realizada a validação do PEP utilizando a abordagem , *Quantitative Evaluation Framework*, que avaliará quantitativamente a ferramenta desenvolvida.

Por último, no Capítulo 10, são apresentadas as conclusões do trabalho desenvolvido e indicadas as linhas de investigação futura para dar continuidade ao trabalho realizado.

2. APRENDIZAGEM DA PROGRAMAÇÃO E MOTIVAÇÃO

Neste capítulo começa por se analisar o processo de ensino/aprendizagem da Programação na tentativa de compreender o insucesso que o caracteriza. Suspeitando-se que o fracasso em causa seja em grande parte decorrente da falta de motivação dos alunos, prossegue-se com um estudo detalhado deste tópico.

2.1 Ensino/ Aprendizagem da Programação

Existem dois conceitos diferentes que geralmente são mal compreendidos pelos alunos (Gomes et al., 1999): a aprendizagem de programação e a aprendizagem da sintaxe e da semântica de uma linguagem de programação. Relembrando o que foi dito acima, programar é, em primeiro lugar, traçar estratégias, a fim de resolver problemas com recurso ao computador, independentemente da linguagem utilizada. De facto, esta tarefa envolve várias etapas desde a compreensão do problema proposto até ao teste do programa, passando pelo desenvolvimento de algoritmos e da sua codificação. Em "*Curriculum Guidelines for Undergraduate Degree Programs in Computer Science*" (ACM/IEEE, 2013), há uma discussão interessante sobre o perigo dos cursos introdutórios de Ciências da Computação serem focados na programação, entendida apenas como codificação. Os autores do referido

documento (cujos princípios são seguidos por Universidades de todo o mundo) alertam: "*Seja ou não a codificação o foco principal de uma disciplina, é importante que os alunos percebem as ciências da computação não apenas como a aprendizagem das especificidades de determinadas linguagens de programação. Devem ser tomados cuidados para realçar os conceitos mais gerais em computação dentro do contexto da aprendizagem*".

Apesar de se considerar que a codificação não é a principal dificuldade, em estudos anteriores (Gomes, 2010) concluiu-se que o paradigma de programação adotado e o idioma utilizado tem um considerável impacto no processo de aprendizagem e, consequentemente, no desempenho da tarefa. No entanto, não existe qualquer consenso entre a comunidade da computação sobre o melhor paradigma ou sobre a linguagem mais adequada; as opiniões divergem. Citando novamente o "*Curriculum Guidelines for Undergraduate Degree Programs in Computer Science"*, acima referido (ACM/IEEE, 2013), os autores sustentam uma importante argumentação sobre o paradigma de programação e escolha da linguagem, acentuando a inexistência de um consenso entre os académicos e recomendando a apresentação de paradigmas de programação alternativos "*para sensibilizar os alunos para as diversas perspectivas na programação*".

Um outro ponto importante que deve ser levado em conta é que aprender a programar é um processo iterativo e incremental. A solução para um problema complexo pode ser obtido em sucessivos passos partindo da resolução de problemas mais simples e assim enriquecer as soluções anteriores. Soluções mais simples são usadas muitas vezes para resolver problemas maiores. Estas técnicas de enriquecimento e composição devem ser consideradas no planeamento das atividades de ensino, conduzindo a proposta em passos incrementais de complexidade crescente.

Seguindo esta abordagem, os alunos podem compreender e adquirir estratégias de resolução de problemas. Por conseguinte, acredita-se que um comportamento ativo, participativo por parte do estudante e que mostre envolvimento e entusiasmo, conduz a uma melhoria da sua capacidade de resolver os problemas propostos. No entanto, os professores percebem que, na maioria dos casos, quando é pedido aos alunos para resolver um problema particular, estes não são capazes de iniciar a tarefa, nem no papel nem no computador. Mesmo quando eles quebram a inércia inicial, muitas vezes torna-se desanimador e desistem facilmente assim que

enfrentam o primeiro obstáculo (Proulx, 2000). Na opinião de alguns professores questionados sobre o assunto, esta afirmação não é verdadeira quando se ensina bons alunos; no entanto, concordam que isto realmente acontece com a maioria dos alunos na generalidade.

2.2 Introdução à Motivação

Várias teorias interdisciplinares de diversas correntes foram desenvolvidas para explicar a motivação desde o inicio da História da Psicologia como ciência. Por se tratar de um fenómeno complexo, multideterminado e com especificidades relativas ao contexto, o assunto tem sido estudado sob diferentes prismas. Por exemplo, algumas teorias afirmam que as pessoas são motivadas por recompensas materiais, outras pelo aumento do seu poder e prestígio no mundo, outras por um trabalho interessante, pelo reconhecimento, ou ainda para ser respeitado como indivíduo. Várias teorias têm surgido, cada uma com o seu contributo para a clarificação do conceito, sendo por isso relevante estudá-las para se atingir uma visão mais abrangente. O facto é que os seres humanos em geral, têm necessidades e desejos muito complexos. A palavra motivação é derivada do verbo motivar e refere-se ao motivo, ou seja, aquilo que move a pessoa (do latim *movere*), que a faz entrar em ação e a leva para algum objetivo (Williams et al., 2011; Almeida, 2012). Está diretamente ligada aos nossos desejos, necessidades e vontades. A motivação é uma das chaves para a compreensão do comportamento humano; age sobre o pensamento, a atenção, a emoção e a ação, envolvendo anseios, desejo, esforço, sonho e esperança.

A representação mais completa da humanidade mostra a pessoa como um ser curioso, vital e auto-motivado. No seu melhor, as pessoas são agentes e inspiradoras, esforçam-se por uma aprendizagem, crescem, dominam novas habilidades e aplicam os seus conhecimentos de forma responsável. Na maioria dos casos as pessoas fazem um grande esforço para se mostrarem corretas, mais do que seres excecionais, exibindo assim algumas características muito positivas e de persistência da natureza humana. Por outro lado, o espírito humano pode ser muitas vezes diminuído rejeitando assim o seu crescimento e as suas responsabilidades, independentemente de extratos sociais ou origem cultural; muitas vezes crianças ou adultos mostram-se apáticos e irresponsáveis. Sendo assim, a natureza humana pode ser ativa ou passiva, construtiva ou destrutiva, e apresenta uma grande variedade de

reações aos diferentes ambientes sociais, sugerindo a existência de diferenças biológicas, psicológicas e educacionais (Williams et al., 2011; Almeida, 2012).

A Teoria da Autodeterminação (SDT – Self- Determination Theory) (Ryan et al., 2000) é uma abordagem à motivação humana e personalidade que usa métodos empíricos tradicionais, destacando a importância da evolução dos recursos internos do ser humano para o desenvolvimento da personalidade e da auto regulação comportamental. O seu foco é a investigação de tendências de crescimento e necessidades psicológicas inatas que serão a base para a sua auto-motivação e integração da personalidade, bem como as condições que promovem esses processos. Usando o processo empírico, foram identificadas três necessidades: necessidade de competência, de relacionamento e de autonomia como representado na figura 2.1.

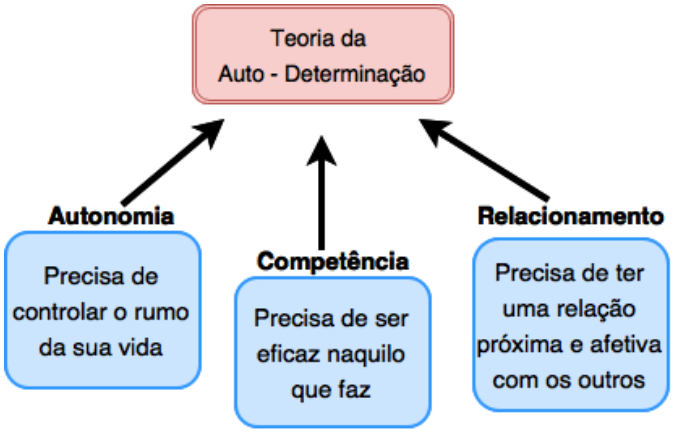


Figura 2.1: Elementos essenciais para a motivação

Deci e Ryan (Ryan et al., 2000), com o objetivo de entender a importância do equilíbrio destas necessidades, fizeram uma analogia com a planta que precisa simultaneamente de água, luz e minerais para se desenvolver; o crescimento, saúde e integridade ficam comprometidos quando qualquer um destes elementos é insuficiente. Também o ser humano necessita de três elementos essenciais para iniciar e manter a sua motivação, que são, a autonomia, a competência e o relacionamento. Estas três necessidades, parecem ser essenciais para facilitar o crescimento saudável e a integração, ou seja, o desenvolvimento social e bem estar pessoal. A motivação diminui quando falha qualquer uma destas necessidades.

A observação sugere que as pessoas são movidas por tipos de fatores muito diferentes, com experiências variadas e respetivas vivências. A motivação conduz a uma ação direcionada a

um determinado objetivo, sendo regulada, biológica ou cognitivamente, pela própria pessoa. Esta ação é ativada por necessidades, emoções, valores, objetivos e expectativas pessoais, constituindo um fenómeno individual, intencional e multifacetado. As pessoas podem ser motivadas porque valorizam uma determinada atividade ou porque há coação externa. Podem ser incentivados por um interesse permanente ou por um suborno, podem-se comportar devido a senso de compromisso pessoal ou porque estão a ser vigiados. Existe diferença entre os casos de motivação interna e motivação externa. Devido às diferenças funcionais entre auto motivação e aos incentivos externos, esta teoria tem a intenção de fornecer uma abordagem mais diferenciada para distinguir os tipos de motivação. Considerando as forças que movem um individuo a agir, esta teoria é capaz de identificar os vários tipos de motivação, os quais têm consequências diferentes para a aprendizagem, desempenho, experiência pessoal e bem-estar (Ryan et al., 2000).

Neste contexto, podem ser considerados dois tipos de motivação: a motivação extrínseca e a motivação intrínseca. A motivação extrínseca é aquela em que a pessoa é movida por condições externas a ela, sejam benefícios ou punições, mas que a ação por si só não a satisfaz. A satisfação não vem da atividade em si, mas sim das consequências extrínsecas produzidas pela atividade. Na motivação intrínseca o que move a pessoa para a ação são motivos internos baseados em necessidades intrínsecas e a gratificação da pessoa é pela ação em si, sem que sejam necessários benefícios externos como impulsionadores. Ou seja, envolve as pessoas numa atividade, porque elas a acham interessante e sentem satisfação espontânea no seu desempenho. A Figura 2.2 ilustra estas ideias. Pois na motivação intrínseca o individuo conhece os seus objetivos pessoais (propósito), tem capacidade de escolher e de traçar caminhos (autonomia) e sabe fazer as suas tarefas (é perito ou especialista).

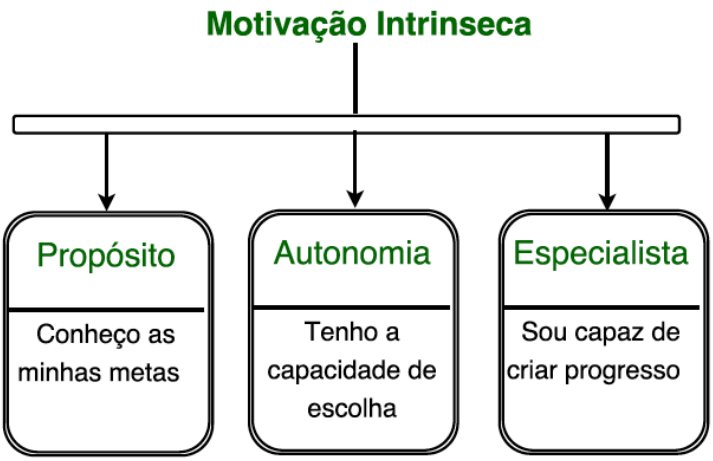


Figura 2.2: Motivação intrínseca

A motivação intrínseca e a extrínseca formam um contínuo, que vai desde a falta de motivação, passando por vários níveis da motivação extrínseca, até chegar à motivação intrínseca, conforme a Figura 2.3 (Ryan et al., 2000).

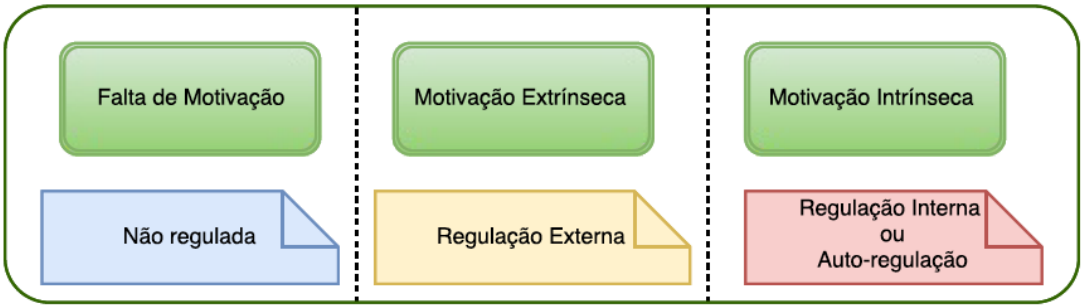


Figura 2.3: Diferentes graus da Motivação

Leal e colegas (Leal et al., 2011) afirmam que a Teoria da Autodeterminação distingue diferentes questões motivacionais: *porquê* versus *para quê*, ou seja qual o objetivo da sua atividade e porque é que a pessoa quer realizar esse objetivo, quais as razões que justificam o seu esforço para atingir esse objetivo (Silva et al., 2010; Leal et al., 2011).

Segundo Campos e Susana Ramos (Campos et al., 2011), existem algumas teorias da motivação, para além da referida Autodeterminação, que caracterizam o indivíduo como único, mas também procuram analisar o fenómeno motivacional na sua origem, evolução e direção. Teorias estas que foram classificadas em *Teorias de Satisfação*, que são orientadas para os fatores que levam a um determinado comportamento motivado, e *Teorias de Progresso*. Como *Teorias de Satisfação* existem a Teoria de Hierarquia das Necessidades de

Maslow de 1954, a teoria X e Y de McGregor de 1960 e a Teoria dos Dois Fatores de Herzberg de 1959. A Teoria das Necessidade Básicas de McClelland de 1961 e a Teoria das Expectativas de Vroom de 1964 são consideradas *Teorias de Progresso*.

A Teoria da Hierarquia das Necessidades de Maslow (Maslow, 1970), diz-nos que a motivação está diretamente ligada à satisfação de necessidades, distinguindo necessidades fisiológicas, necessidades de relação e necessidades de ser. Estas necessidades estão arranjadas segundo uma hierarquia que Maslow denominou hierarquia dos motivos humanos. Uma necessidade é substituída pela seguinte de ordem superior na hierarquia, sempre que esta for satisfeita, de tal modo que quando uma necessidade é satisfeita, outra ocupa o seu lugar em busca da satisfação. A necessidade fisiológica é, portanto, a mais forte, a mais básica ou de ordem inferior porque têm a ver com a preservação e conservação da espécie, enquanto a necessidade de auto-realização é a mais fraca, de necessidade secundária ou de ordem superior porque se relaciona com a identidade psicossocial.

Esta proposta de hierarquia de tipos de necessidades é conhecida como Pirâmide de Maslow que nos diz que na base da pirâmide existem necessidades indispensáveis à vida, chamadas de necessidades primárias, como a satisfação de necessidades fisiológicas básicas do corpo (respiração, comida, água, etc.); Logo acima vem a necessidade de segurança, de amor e relacionamentos e de estima; por último, no grau mais refinado, vêm as necessidades de realização pessoal, chamadas de necessidades secundárias. Sendo assim, poucas pessoas são privilegiadas e conseguem chegar ao topo da pirâmide e serem fortemente motivadas pelas necessidades de nível mais alto.

Uma das críticas feitas à teoria refere-se ao facto de que nem sempre existe uma hierarquia das necessidades tão bem definida e genérica, podendo certa pessoa dirigir-se diretamente a necessidades superiores, ou seja, pode haver algumas necessidades, que para uma determinada pessoa ou sociedade não tenham a mesma importância. Algumas necessidades da hierarquia são, muitas vezes, valores (que podem desaparecer ou modificar) e não necessidades (inatas e universais).

A Pirâmide de Maslow, no entanto, tornou-se um importante modelo explicativo da Psicologia Organizacional (Campos et al., 2011). Leituras complementares em [1].

A Teoria X e Y de McGregor (McGregor, 1960) apresenta dois vetores diametralmente opostos, defendendo a ideia de que existem dois tipos fundamentais de pessoas, dois perfis de personalidade e comportamento, chamados por ele dimensões X e Y. A primeira (Teoria X) trata as pessoas, no geral, como preguiçosas, para quem todo o trabalho é visto como um mal necessário; neste caso, toda a lógica da organização vai-se voltar ao desenvolvimento de processos que induzam as pessoas a produzirem. A segunda (Teoria Y) parte do pressuposto contrário de que as pessoas, além da necessidade de trabalhar, gostam de fazer os seus ofícios diariamente, ou seja, o trabalho é tão natural como o lazer, se as condições forem favoráveis. Os comportamentos das pessoas são então divididos em comportamentos do tipo X (baseados na motivação extrínseca) e em comportamentos do tipo Y (baseados na motivação intrínseca).

A Teoria dos Dois Fatores de Herzberg (Herzberg, 1968) aborda a situação de motivação e satisfação das pessoas. O objetivo é entender os fatores que causam insatisfação e aqueles que são os responsáveis pela satisfação no ambiente de trabalho. Há duas classes distintas de fatores que condicionam o comportamento e o grau de satisfação: a primeira classe chama-se de fatores higiénicos e relacionam-se com fatores extrínsecos como as estratégias administrativas, estilo de liderança, as relações interpessoais, as condições económicas e ambientais; e a segunda de fatores motivacionais que se relacionam com fatores intrínsecos como realização, reconhecimento, responsabilidade, trabalho em si e possibilidade de progressão na carreira. Os fatores que conduzem à satisfação são separados e diferentes dos que determinam a insatisfação (Ramos, 2013). Leitura relacionada com esta teoria pode ser consultada em [1].

A Teoria das Necessidade Básicas de McClelland (McClelland, 1967), como teoria de progresso, é um modelo motivacional que tenta explicar como é que três necessidades básicas (realização, poder e afiliação) interferem nas ações das pessoas a nível organizacional e de gestão. Essas necessidades são desenvolvidas pelo individuo a partir da sua experiência

[1] http://www.esoterikha.com/coaching-pnl/o-que-e-motivacao-conceito-e-principais-teorias-da-motivacao-definicao.php

de vida e das suas interações com outros indivíduos e com o ambiente, independentemente do género, cultura ou idade. Cada pessoa tem uma destas necessidades mais dominante do que as outras duas, o que pode depender da cultura e experiências de vida. A necessidade de realização traduz o desejo de alcançar algo difícil, atingir objetivos que representem desafios de fazer da melhor maneira e mais eficientemente (o seu maior desejo é ser excelente), fazer alguma coisa melhor do que anteriormente. A necessidade de afiliação é o desejo de estabelecer relacionamentos pessoais próximos, evitar conflitos e estabelecer fortes amizades. A amizade e o bom relacionamento social são importantes para estas pessoas. Por último, a necessidade de poder traduz-se pelo desejo de influenciar ou controlar outros, ser responsável por outros e ter autoridade sobre os outros. São pessoas que procuram posições de liderança (Campos, 2011). Leituras relacionadas com a teoria de McClelland, podem ser vistas nos endereços [2,3,4].

Por outro lado, a Teoria das Expectativas de Vroom (Vroom, 1964) explica como é que o individuo decide qual o comportamento a tomar para a realização do seu trabalho. Vroom defende que a força da motivação (M) de determinada pessoa corresponde ao produto do valor atribuído a um objeto (V-Valencia) pela probabilidade de alcançar esse mesmo objetivo (E-Expectativa):

$$M=V \times E.$$

Assim sendo, a motivação é nula quer no caso em que o valor do alvo é zero, ou seja, é indiferente atingir ou não determinado objectivo, quer no caso em que não existe qualquer expectativa em atingir o resultado (a probabilidade de o alcançar é zero). Da mesma forma, ocorre desmotivação sempre que a valência é negativa, isto é, quando a pessoa prefere não atingir o objectivo. De acordo com esta teoria, diferentes pessoas reagem de maneira diferenciada perante uma situação (Campos, 2011). Para mais leituras relacionadas ver [5].

Depois de uma introdução geral à Motivação humana---sua complexidade, fatores que a determinam e impacto que tem no comportamento das pessoas---e depois de apresentadas

[2] https://www.portal-gestao.com/artigos/7391-teoria-das-necessidades-de-mcclelland.html

[3] http://www.esoterikha.com/coaching-pnl/teoria-das-necessidades-adquiridas-teoria-de-macclelland-motivacao.php

[4] http://pt.slideshare.net/RonneSeles/a-teoria-das-necessidades-de-david-mc-clelland

[5] http://www.okconcursos.com.br/apostilas/apostila-gratis/111-administracao-geral/159-teorias-da-motivacao#.V_0Pe4S8wmU

as principais teorias desenvolvidas essencialmente no âmbito da motivação para as atividades profissionais (psicologia do trabalho, ou empresarial), vai-se estudar na próxima secção a motivação na perspetiva dos alunos.

2.3 Motivação dos Alunos do Ensino Superior

Motivar os alunos é um dos maiores desafios que os professores têm que enfrentar. Mas a verdade é que os professores apenas têm controle sobre os fatores externos que influenciam o comportamento e envolvimento dos alunos, para poder desempenhar um papel vital na formação que ocorre em sala de aula. Porém, todas as decisões de ensino escolhidas têm um impacto (positivo ou negativo) sobre a motivação dos alunos (Callahan, 2010), pelo que importa perceber quais aquelas que podem afetar positivamente.

O grau de motivação em que um aluno se encontra para se envolver numa determinada tarefa é determinada em conjunto pela sua expectativa para o sucesso e pelo valor que ele dá a uma tarefa específica. Esta teoria sugere que os estudantes podem ter êxito se se aplicarem com esforço e apreciarem o valor das atividades em que se envolvem, como referido na Teoria de Vroom apresentada na secção anterior.

Importa perceber por que é que os alunos não têm motivação. Muitos alunos atribuem esse problema aos comportamentos do professor e à escola em geral, tendo a expectativa de que o professor seja um elemento ativo na sua aprendizagem. Por outro lado, o professor atribui as dificuldades na sala de aula aos próprios alunos, tendo a expectativa de que estes sejam interessados, autorregulados, que tenham energia para procurar o conhecimento e que sejam responsáveis pela sua própria motivação. Neste sentido, parece haver conflito entre alunos, que chegam com certas expectativas, e professores, que esperam deles comportamentos diferentes daqueles que, em geral, eles manifestam (Almeida, 2012).

Segundo Almeida (Almeida, 2012), uma visão mais precisa de motivação, consiste em considerar que a motivação não é apenas um fenómeno unitário que remete ao conceito de quantidade. Mais do que quantidade de motivação, existem variações nos níveis e nas orientações motivacionais. Sendo assim, é possível perguntar qual será o factor que leva a um comportamento mais ou menos motivado. Ao pensar em qualidade motivacional consideram-se as atitudes e metas que dão motivo para a ação, isto é, na razão das ações. Um bom

exemplo é o motivo que se tem para realizar tarefas. Um aluno que está a fazer os trabalhos de casa pode estar sem qualquer curiosidade ou interesse no trabalho em si, mas ser cativado pelo fato de querer a aprovação do professor ou dos pais; mas pode pelo contrário estar motivado para adquirir novos conhecimentos e novos desafios porque entende que isso lhe traz vantagem e o valoriza; ou pode ainda estar motivado porque os conhecimentos adquiridos lhe dão condições de ter boas notas e uma vida social melhor. Nota-se, neste exemplo, que a motivação pode não variar quantitativamente, mas a sua natureza pode ser definitivamente distinta. Distinguir aspectos quantitativos e qualitativos da motivação permite ampliar a visão sobre ela, conforme se vê na Figura 2.4.

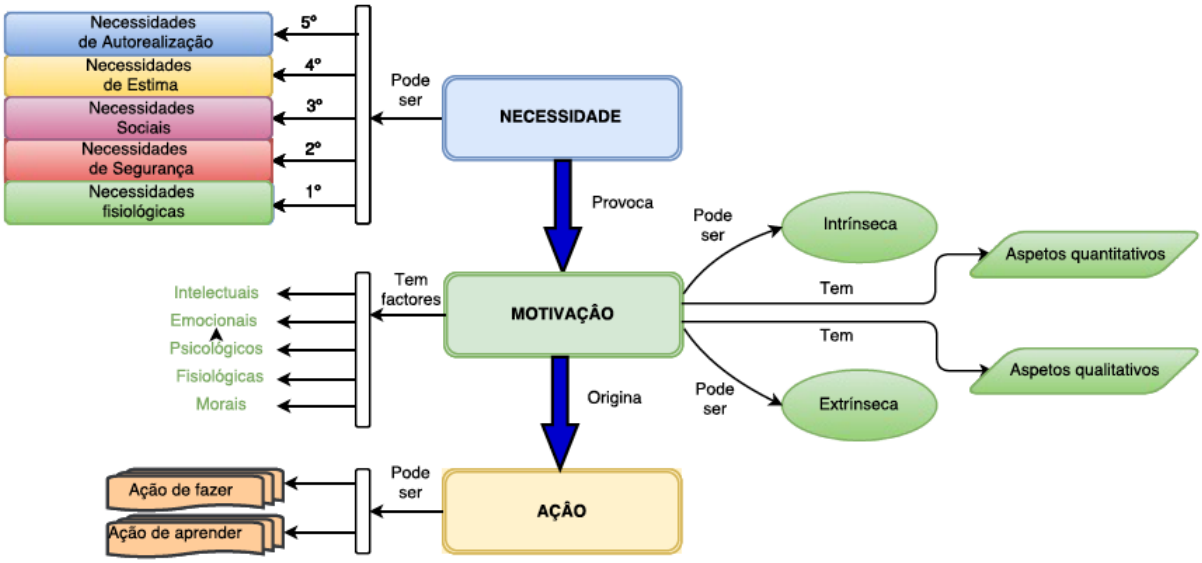


Figura 2.4: Fatores envolvidos na motivação e seu impacto

Na procura da realização de objetivos e metas, o aluno necessita de motivação que atua como impulsionado fazendo com que estes dêem o melhor de si (Silva et al., 2014). Essa motivação é referida às forças dentro do individuo responsáveis pela direção e pelo nível de persistência do esforço gasto em determinado trabalho. Esforço esse determinado pelo aluno na escolha do que fazer. Assim, a motivação está diretamente relacionada com fatores de ordem psicológicos, fisiológicos, morais, intelectuais e emocionais (ver Figura 2.4).

A motivação do aluno é um dos componentes essenciais para a aprendizagem. Sendo assim, qual a razão que leva um aluno a ter um comportamento mais ou menos motivado? Como explicado no parágrafo acima, várias podem ser essas razões: apenas interesse de ser aprovado, desprovido de curiosidade pelo tema; interesse em adquirir conhecimento;

interesse nos benefícios diretos ou indiretos que esses novos conhecimentos lhe podem trazer, como obter boas notas ou arranjar um bom emprego. De acordo com a teoria de autodeterminação, essa motivação pode ser intrínseca ou extrínseca. A motivação intrínseca não precisa de qualquer fator externo, tem origem no próprio aluno, como por exemplo a dedicação, a competência, a vontade e apetência para realizar uma tarefa. A motivação extrínseca, é resultante de fatores externos, como os recursos que o aluno tem, as recompensas ou penalizações, e o ambiente onde desenvolve as suas tarefas. Ambas atuam em conjunto e o resultado vai definir o comportamento do aluno, como apresentado na Figura 2.5. O que é certo é que os fatores motivacionais e afetivos têm relação direta com a aprendizagem (Silva et al., 2014).

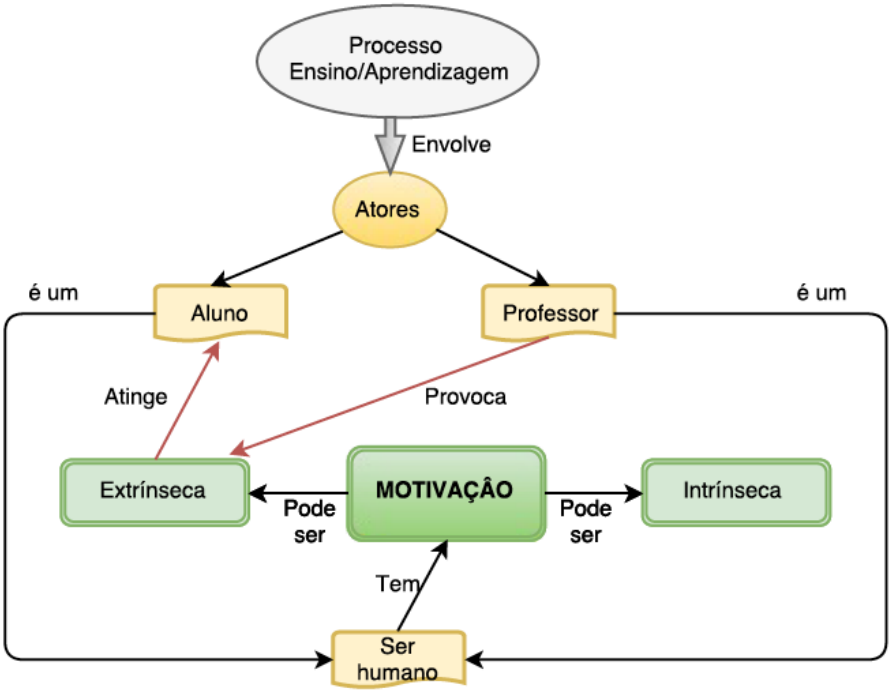


Figura 2.5: Motivação Aluno/Professor

2.4 Estímulos para a Motivação no Ensino

Como referido nas secções anteriores, é muito importante que os professores consigam melhorar o processo da aprendizagem atuando o mais eficazmente possível nos fatores extrínsecos da Motivação. Os alunos são complexos e têm necessidades e desejos bastante diversos e sobretudo distintos. Segundo Kaylene e Caroline Williams (Williams et al., 2011),

existem cinco fatores essenciais a ter em consideração para melhorar a motivação dos alunos, que são:

- O próprio aluno
- O professor
- O conteúdo da matéria
- O método/processo de ensino
- O ambiente de aprendizagem

Os autores referem, por exemplo, que o aluno deve ter acesso à informação, capacidade de raciocínio e interesse por todo o processo educativo em que está inserido. O professor deve ser bem treinado, concentrar-se e monitorizar todo o processo, ser dedicado e inspirador. Por sua vez, o conteúdo deve ser preciso, oportuno, estimulante e pertinente para o conhecimento atual e necessidades futuras do aluno. O quarto ingrediente, método/processo, deve ser inventivo, encorajador, interessante, envolvente, e fornecer ferramentas que podem ser aplicadas à vida real do aluno. E por último, o ambiente deve ser acessível, seguro, positivo e o mais personalizado possível (Williams et al., 2011).

Manter os alunos motivados é um objetivo particularmente importante em disciplinas de introdução à programação nos cursos de engenharia. Para que essa motivação seja impulsionada, os alunos devem aumentar a sua competência na resolução de problemas, uma capacidade que é construída sobre um domínio combinado de conceitos e competências. Esta combinação de conceitos e habilidades requer uma preparação didática e pedagógica que conduza os alunos a aprenderem mais do que a soma dos conceitos ensinados e adquiram novas capacidades. Quer dizer que um conjunto representativo de conceitos deve ser incorporado em todo o processo para orientar os alunos através de níveis crescentes de "competências computacionais". A aprendizagem baseada em problemas apoia melhor este processo, porque os alunos entram em contacto com os conceitos através das suas próprias atividades e, assim, podem melhor diferenciá-los. Esta diferenciação estabelece o terreno para uma percepção das ideias subjacentes que permite aos alunos construir os conceitos por si mesmos, aplicá-los com êxito enquanto estão no controle de seu próprio processo de aprendizagem (Faessler et al., 2006). Um ponto crucial, no entanto, é que os problemas que orientam este processo devem ser interessantes, relevantes, realistas e, se possível, também

divertidos. O nível de dificuldade dos problemas deve aumentar durante as diferentes fases de aprendizagem. Começar com pequenos problemas que incorporam um conjunto mínimo de conceitos continuando com problemas mais complexos. Estes são os ingredientes preliminares para a instrução que motiva. Para isso, é muito importante o docente envolver-se e apoiar na resolução de problemas desde o início, em conjunto com uma construção auto-dirigida de estruturas de conhecimento para o aluno que seja profundo e sustentável. Resolver uma sequência de pequenos problemas selecionados representando conceitos fundamentais leva a um aumento da motivação, desde que acompanhados de tarefas ambiciosas, em vez de triviais (Faessler et al., 2006).

Segundo Faessler (Faessler et al., 2006), a aprendizagem é mais eficaz e eficiente se se tornar este processo de aprendizagem amigável à aquisição de conhecimento dividindo-o em quatro passos discretos:

- Ver: os alunos devem ter oportunidade de tomar contacto com os conceitos;
- Tentar: os alunos devem ter a oportunidade de tentar aplicar conceitos com a orientação apropriada;
- Fazer: os alunos devem ser capazes de aplicar os conceitos de forma independente, autónoma;
- Explicar: para verificar a sua aprendizagem, os alunos devem explicar a sua solução ao professor.

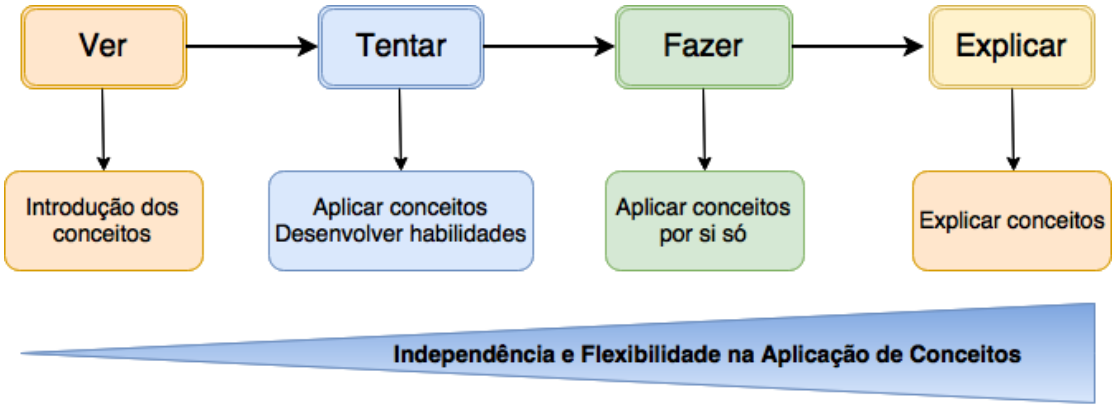


Figura 2.6: Conjunto de passos para um processo de aprendizagem amigável

Durante o segundo passo (Tentar), podem ser usados métodos construtivistas em ambientes de ensino apoiado em computador para permitir uma melhor orientação individualizada (Faessler et al., 2006).

Reforçando o que foi dito anteriormente, sistematiza-se agora como é que o professor pode aumentar a motivação dos seus alunos de acordo com outros autores da área educativa (Taylor et al., 2011):

- Doseando progressivamente a dificuldade dos problemas e o tempo que a sua resolução requer, pois, o aluno motiva-se ao sentir-se capaz de resolver um problema em tempo útil;
- Evitando exercícios e tarefas rotineiras;
- Introduzindo desafios adaptados para cada turma atendendo às características de cada curso e comportamento/atitude dos alunos;
- Dando *feedback*, quer de afirmações/dúvidas, quer dos exercícios resolvidos;
- Incentivando a participação e apelando a um espírito crítico.

Os alunos mudaram muito nos últimos vinte anos, como resultado de uma educação rica em tecnologia. A evolução tecnológica das últimas décadas tem trazido para a população em geral um crescente número de recursos que gradualmente têm modificado os hábitos e formas de estar, fazer e aprender. É claro que os alunos atuais manifestam necessidades, objetivos e preferências de aprendizagem diferentes comparativamente com os estudantes no passado. Por este motivo, é importante compreender estes alunos para determinar como melhor envolvê-los na aprendizagem (Taylor et al., 2011). O uso das tecnologias educativas, uma vez que influenciam e potenciam o processo de aprendizagem, tem vindo a refletir-se nas estratégias didáticas no contexto atual. Neste sentido, tem-se desenvolvido diferentes propostas para promover a educação; a gamificação é um desses exemplos (Valentim et al, 2016).

2.4.1 Gamificação

Gamificação (do inglês, *gamification*), ou mais apropriadamente em português a Ludificação, é o principio que consiste em aplicar elementos de jogo em contextos da vida real como, por exemplo, na educação, com o objectivo de criar mudanças comportamentais, aumentando a

motivação e o desempenho dos indivíduos. Sendo assim, gamificação não é apenas o uso de jogos na edução, mas a utilização de elementos relacionados com os jogos criando uma narrativa diferenciada no processo educativo, permitindo aprendizagem através dos desafios, pelo prazer e entretenimento[6] (Valentim et al, 2016).

Com o aparecimento deste conceito, têm sido propostas muitas definições para gamificação. Este grande número de definições mostra que não existe ainda um consenso sobre o que é a gamificação e que a comunidade científica e os profissionais que estão a aplicar o conceito ainda estão a procurar os seus fundamentos teóricos (Simões, 2015).

Embora haja um aumento da sua utilização, muitas tentativas falharam os seus objetivos, por má abordagem prática e por opiniões controversas. No entanto, teorias motivacionais, como a teoria da autodeterminação ou a teoria do estado de fluxo, termo português usado vulgarmente para designar The Flow-Theory proposta por Cziksentmihalyi no seu livro (Csikszentmihalyi, 1975) (que explica a evolução do *estado emocional* de um aluno entre a *ansiedade* e o *aborrecimento*) podem servir como valiosas bases para compreender como é que o recurso a técnicas e elementos dos jogos podem influenciar a motivação humana[6]. É muito importante que os jogos mantenham um equilíbrio entre a ansiedade e o aborrecimento aplicando elementos presentes na teoria do estado de fluxo, ou seja, que a atividade de um jogo não seja demasiado difícil ou fácil de realizar. No contexto da gamificação, este equilíbrio é conseguido através de níveis de progressão de dificuldade e do feedback instantâneo (Monteiro and Santos 2014).

Mihaly Cziksentmihalyi pretende estudar o envolvimento das pessoas nas tarefas que têm de realizar (neste caso, nos jogos) considerando que o topo, ou pico desejável, dessa entrega corresponde a "um estado emotivo em que uma pessoa está tão envolvida numa atividade que mais nada parece importar; a experiência é tão agradável que a pessoa continua a fazê-lo mesmo a grande custo, pelo simples motivo de o fazer (Cskikszentmihalyi, 1990), procurando explicar com sua teoria como é que a intensidade deste estado vai fluindo ao longo do tempo. É um estado que leva a que uma pessoa ao estar tão envolvida numa atividade se abstrai do tempo e do espaço, procurando transportar os elevados níveis de envolvimento e de

[6] http://enterprise-gamification.com/index.php?option=com_content&view=article&id=280:master-thesis-gamification-epic-win-or-epic-fail&catid=4&Itemid=251&lang=en

motivação (sentidos pelos jogadores) para contextos diferentes dos jogos (por exemplo para contexto de aula). Para que seja possível um envolvimento, é importante existir um percurso, um canal de fluxo definido por uma sequência de objetivos. Esse canal onde o fluxo se desenvolve é limitado pela ansiedade (não é capaz de realizar a tarefa) e pelo aborrecimento (tarefa menos desafiante). À medida que as competências se desenvolvem, o desafio tende a afastar-se da zona da ansiedade para a zona de aborrecimento (Simões, 2015).

Neste contexto, a gamificação tem como proposta a concretização da interatividade e aprendizagem colaborativa a partir de novas narrativas educativas, ou seja, traz um novo significado à prática educacional, na medida que promove o desenvolvimento de uma nova dinâmica de aprendizagem assente no envolvimento dos alunos através do uso de elementos de jogos em contexto pedagógico. Nem todas as situações são passíveis de serem gamificadas, uma vez que dependem da natureza do objeto de conhecimento a ser ensinado, das competências e habilidades a desenvolver nos estudantes. Por outro lado, se a gamificação se tornar uma atividade quotidiana, corre-se o risco de todo o processo se tornar cansativo, reduzindo-o à competitividade e ao sistema de recompensas. No ensino Superior, a gamificação pode contribuir para o processo de construção de conhecimento. O aluno utiliza elementos dos jogos, como mecânica, estratégia e pensamento, fora do contexto dos jogos, ampliando a possibilidade de motivar a ação, auxiliar na solução de problemas e promover a aprendizagem. Permite, assim, um cenário interativo mais motivador e permite uma aprendizagem de modo colaborativo, dinâmico e significativo. A aplicação de técnicas de gamificação pode contribuir para transformar uma tarefa monótona num processo de aprendizagem que se torne viciante para os alunos. Para os estudantes, a gamificação pode servir para minimizar o impacto negativo que podem encontrar nas formas tradicionais de educação e os professores podem enfatizar melhor os seus objetivos (dando mais relevo aqueles que achem primordiais) e verificar mais facilmente o envolvimento e desenvolvimento dos seus alunos (Valemtim et al, 2016; Almeida, 2016).

Como explica Ana Klock e colegas (Klock et al., 2014), para se perceber a gamificação, é importante perceber o funcionamento de um jogo, em que as suas atividades estão orientadas a objetivos e metas, assim como a informação clara das condições para se chegar à vitória e dos seus obstáculos. Os jogos baseiam-se essencialmente em três componentes:

atividades focadas em metas; progresso do jogador; e mecanismos de recompensa. Assim, através de pontuações e níveis de experiencia, jogadores ou alunos são levados a completar tarefas para atingir diferentes objetivos, para vencer um jogo ou tirar uma boa nota na sua avaliação escolar. Neste processo, é importante seguir o trajeto do ator (aluno ou jogador) para se poder identificar as tarefas e metas para que este chegue a um determinado nível de aprendizagem ou à vitória (em ambiente de jogo). Para se medir este progresso utilizam-se os mecanismos de recompensa assim como no processo educacional, o *feedback* (verifica se o método da gamificação está a atingir o seu objetivo, ou seja, a aprendizagem), e quanto mais rápido e orientado for esse, mais eficaz será a aprendizagem. Sistematizando, os principais elementos de jogo usados na gamificação são:

- **Pontos**: Este elemento é aberto, direto, e motivacional. Pode utilizar-se vários tipos diferentes de pontuação, de acordo com o objetivo proposto;
- **Níveis**: Estabelecem um escalonamento na dificuldade das várias subtarefas, permitindo ir controlando o progresso do jogador;
- **Rankings**: Comparam os jogadores envolvidos, ou seja, mostram a progressão dos jogadores dentro do ambiente gerando a competição entre eles;
- **Missões**: Orientam os jogadores sobre as atividades que devem ser realizadas, ou seja, estabelecem os desafios que devem ser concluídos;
- **Medalhas**: Representação visual de alguma realização ou conquista do utilizador (versão mais robusta que os pontos);
- **Barra de progresso**: Representação visual da evolução temporal do jogador no cumprimento da sua tarefa;
- **Personalização**: Possibilidade que o utilizador tem de transformar alguns elementos a seu gosto promovendo motivação, envolvimento, sentimento de posse e controlo sobre o sistema;
- **Feedback**: fornece dados ao jogador, informando a sua localização no ambiente e os resultados das ações realizadas por ele;
- **Regras**: Definem como o jogador pode utilizar o ambiente, como funciona, o que é permitido, etc;
- **Narrativas**: Definem um contexto ou história enquadrando os desafios propostos para aumentar o envolvimento do jogador;

- **Partilhas**: dar, solicitar ou trocar recompensas recebidas;
- **Convites**: Chamar outros a participar numa tarefa.

Com todos estes elementos, é possível definir as estratégias/mecânicas do jogo combinando-os de modo a atingir os diferentes objetivos: Comunicação e recompensa; envolvimento social; e experiência do jogo (Figura 2.7).

Figura 2.7: Objetivos Lúdicos versus Elementos de jogo

Como estes elementos são motivadores extrínsecos, o seu efeito e aplicação são limitados. As recompensas extrínsecas devem ser usadas para tarefas e atividades para as quais as pessoas não estão intrinsecamente motivadas para realizar (são importantes para comunicar resultados (*feedback*) mas não são a melhor escolha como estratégia de longo prazo) porém, não substituem a importância da motivação intrínseca; têm um efeito pontual e mais superficial. Se um utilizador executa uma tarefa pela própria tarefa, isso significa que está intrinsecamente motivado para a realizar e assim executa-a melhor. A gamificação, no âmbito da educação, deve criar uma experiência que não dependa apenas de recompensas extrínsecas (Simões et al., 2013).

É importante salientar que alguns métodos podem apenas funcionar num determinado contexto em que se insere e para um determinado público alvo, e não garantir o sucesso noutros. Na verdade, não existem modelos de gamificação que resultem bem em qualquer situação (Simões et al., 2013).

Ana Klock e colegas (Klock et al., 2014) analisaram várias técnicas de gamificação existentes para perceber quais as utilizadas em alguns Ambientes Virtuais de Aprendizagem (AVAs), que

levam a um maior envolvimento e motivação. Da análise de dez AVAs, seis elementos de jogo existentes na literatura e acima listados estão presentes nos ambientes estudados utilizando diferentes estratégias. Concretamente os AVAs considerados nesse estudo foram: Khan Academy, PeerWise, QizBox, BrainScape, Peer2PeerUniversity, URI Online Judge, CodeSchool, Duolingo, Passei Direto e MeuTutor. Para cada ambiente mencionado foi feita uma análise para encontrar as respetivas técnicas e elementos de jogo. A partir dos dados retirados da análise de cada ambiente, foram construídos gráficos para mostrar mais facilmente quantos e quais elementos de gamificação foram aplicados em cada ambiente conforme se ilustra nas Figuras 2.8 e 2.9.

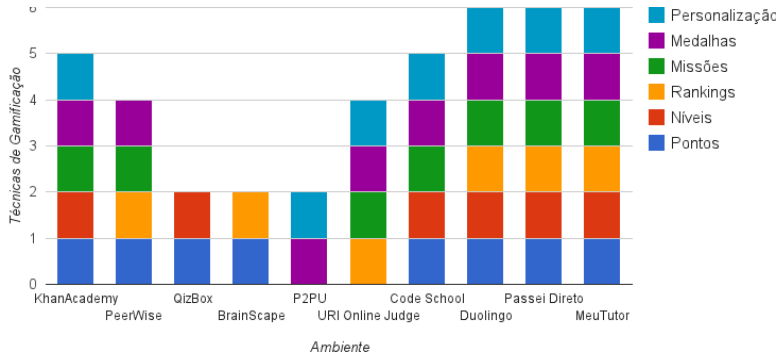


Figura 2.8: Técnicas de Gamificação por AVA's (Klock et al., 2014)

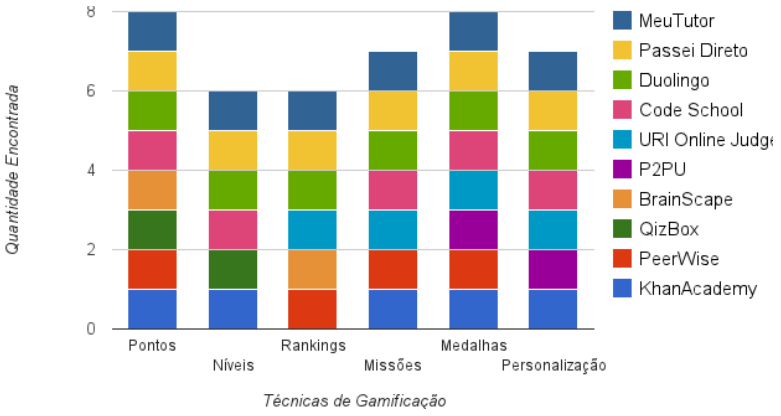


Figura 2.9: Técnicas de Gamificação mais aplicadas (Klock et al., 2014)

Assim, a citada autora concluí que os elementos de jogo mais utilizados os vários e importantes AVAs estudados foram: Pontos e Medalhas; Missões e Personalização; e Níveis e Rankings. Como se pode constatar, diferentes ambientes permitem a inclusão de vários elementos de jogo.

O conceito de gamificação pode ser simples mas tornar um sistema gamificado pode não o ser. Para isso devem seguir-se alguns passos como explicitado na Figura 2.10:

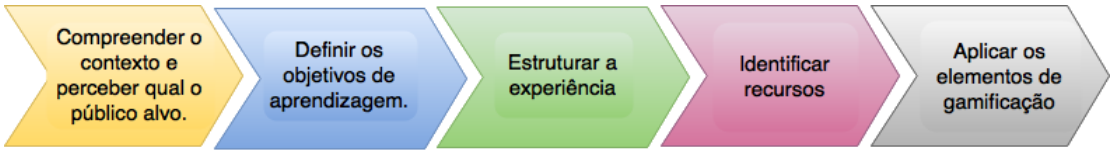


Figura 2.10: Passos para desenvolver um Sistema Gamificado

O primeiro passo, passa por perceber quem é o público-alvo (idade, capacidade de aprendizagem, etc.) e qual o contexto em que se desenvolve o ensino (ambiente envolvente, tamanho dos grupos, duração, etc.). A combinação entre aluno e contexto permite a concepção de uma ferramenta informática que ajuda o aluno a melhor atingir os objetivos do curso. O segundo passo diz respeito ao objetivo que o professor quer que o aluno aprenda e supere ao completar o curso. Embora alguns programas de aprendizagem possam englobar vários objectivos diferentes de uma só vez, o sucesso da ferramenta depende da capacidade do professor de definir claramente os objectivos subjacentes ao programa educativo. No terceiro passo, os professores devem quantificar o que os alunos precisam de aprender e alcançar no final de cada etapa ou da atividade. Esta quantificação também é importante para os alunos, pois torna o objetivo final mais acessível e mensurável e permite identificar todos os obstáculos dentro e entre cada etapa. O professor deve iniciar a sua atividade com etapas mais fáceis para que o aluno se sinta motivado e envolvido. Identificar recursos é o quarto passo e uma vez que as etapas já foram identificadas, o professor pode escolher aquelas que podem ser gamificadas e como. O professor pode usar níveis dentro das etapas associando-lhes recompensas e definindo regras para receber e dar *feedback*. O *feedback* é como se sabe um aliado importante no processo ensino aprendizagem, podendo ser dadas outras oportunidades para completar a tarefa visto que os alunos aprendem mais quando repetem. Isto é o que torna os jogos atraentes. Como aos alunos é dado *feedback* rápido, se eles fizerem uma tarefa errada podem ter a oportunidade de experimentá-la novamente. Além disso, o tempo que o aluno consome para resolver a tarefa, é importante para dar ao professor informação sobre o conhecimento que o aluno vai adquirindo ou sobre os seus pontos fracos. Por último, o quinto passo, consiste em saber que elementos de jogo devem ser aplicados. Como já foi dito, elementos como pontos, medalhas, níveis, restrições de tempo, etc., fazem com que os alunos se concentrem em competir consigo mesmos e

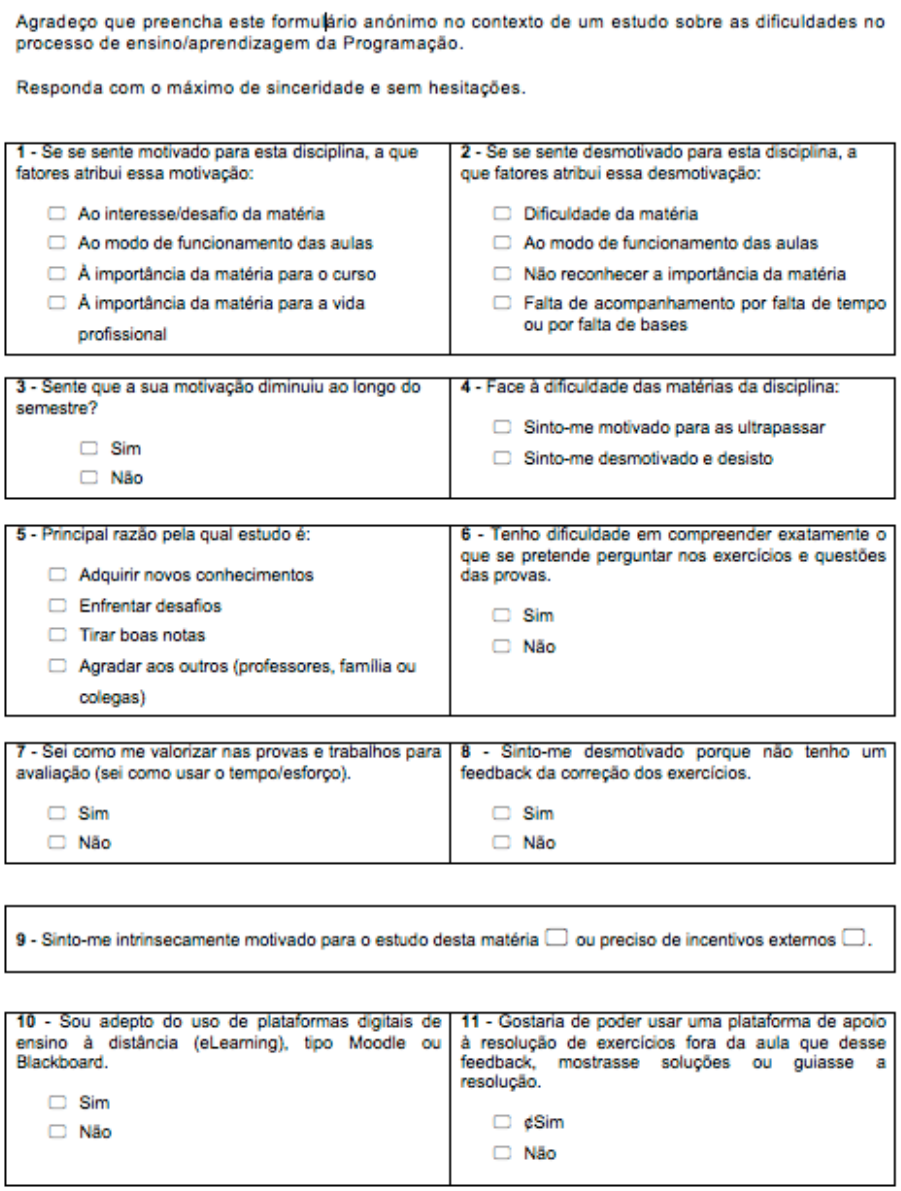
reconhecer a auto-realização. Por outro lado, existem elementos especiais que para explorar a competição interativa, como por exemplo as tabelas de classificação, permitem aos alunos jogar uns com os outros partilhando o seu progresso e conquistas. É importante recordar que um tipo específico de elementos pode desencadear reações diferentes nos alunos e quando não é usado corretamente, pode ser contraproducente (o aluno pode sentir-se intimidado em aprender algo novo) (Huang et al., 2013).

Segundo os autores (Huang et al., 2013), os professores devem passar por estas cinco etapas de gamificação na educação (esquematizadas na figura 2.10 acima) para conseguirem implementar os diferentes elementos de gamificação em programas de aprendizagem e alcançar os objetivos pretendidos. Se os objetivos forem claros, o contexto ajudará a determinar os pontos fracos. A precisão e a eficiência da aplicação da gamificação ao programa de ensino dependerão do rigor da implementação destas cinco etapas.

Depois de se ter introduzido o conceito de Motivação e se ter falado nas teorias desenvolvidas à sua volta de forma genérica, depois de se ter falado em particular na motivação dos alunos e após se terem analisado estratégias (nomeadamente a gamificação) para estimular a motivação dos alunos, discute-se na próxima secção e antes de fechar este capítulo um inquérito feito a alunos de programação de vários cursos e várias instituições para perceber o que pensam eles próprios sobre a sua motivação para esta disciplina.

2.5 Inquérito: como sentem os alunos a motivação

De forma a aferir as reais dificuldades que os alunos sentem no processo de ensino/aprendizagem da programação, foi elaborado um inquérito, como se mostra abaixo na Figura 2.11, que numa primeira instância, foi aplicado aos alunos de uma disciplina de primeiro ano algoritmia e programação.



Agradeço que preencha este formulário anónimo no contexto de um estudo sobre as dificuldades no processo de ensino/aprendizagem da Programação.

Responda com o máximo de sinceridade e sem hesitações.

1 - Se se sente motivado para esta disciplina, a que fatores atribui essa motivação:

- ☐ Ao interesse/desafio da matéria
- ☐ Ao modo de funcionamento das aulas
- ☐ À importância da matéria para o curso
- ☐ À importância da matéria para a vida profissional

2 - Se se sente desmotivado para esta disciplina, a que fatores atribui essa desmotivação:

- ☐ Dificuldade da matéria
- ☐ Ao modo de funcionamento das aulas
- ☐ Não reconhecer a importância da matéria
- ☐ Falta de acompanhamento por falta de tempo ou por falta de bases

3 - Sente que a sua motivação diminuiu ao longo do semestre?

- ☐ Sim
- ☐ Não

4 - Face à dificuldade das matérias da disciplina:

- ☐ Sinto-me motivado para as ultrapassar
- ☐ Sinto-me desmotivado e desisto

5 - Principal razão pela qual estudo é:

- ☐ Adquirir novos conhecimentos
- ☐ Enfrentar desafios
- ☐ Tirar boas notas
- ☐ Agradar aos outros (professores, família ou colegas)

6 - Tenho dificuldade em compreender exatamente o que se pretende perguntar nos exercícios e questões das provas.

- ☐ Sim
- ☐ Não

7 - Sei como me valorizar nas provas e trabalhos para avaliação (sei como usar o tempo/esforço).

- ☐ Sim
- ☐ Não

8 - Sinto-me desmotivado porque não tenho um feedback da correção dos exercícios.

- ☐ Sim
- ☐ Não

9 - Sinto-me intrinsecamente motivado para o estudo desta matéria ☐ ou preciso de incentivos externos ☐.

10 - Sou adepto do uso de plataformas digitais de ensino à distância (eLearning), tipo Moodle ou Blackboard.

- ☐ Sim
- ☐ Não

11 - Gostaria de poder usar uma plataforma de apoio à resolução de exercícios fora da aula que desse feedback, mostrasse soluções ou guiasse a resolução.

- ☐ ¢Sim
- ☐ Não

Figura 2.11: Inquérito feito a 160 alunos da disciplina da Algoritmia e Programação

Nesta primeira instância, o inquérito foi realizado de forma anónima no dia do exame da disciplina e aplicado a um total de 160 alunos, tendo-lhes sido pedido para responderem com o máximo de sinceridade e sem hesitação. Mais tarde foi usado noutras Instituições de Ensino Superior com alunos de programação em disciplinas diversas, conforme se falará na subsecção seguinte.

Em primeiro lugar observou-se que todos responderam ao inquérito em geral e que a percentagem de respostas em branco a cada questão foi sempre inferior a 2%. Algumas dos principais resultados às questões mais críticas podem ser apreciados na Tabela 2.1.

	Sim	Não	Não respondeu
Sente que a sua motivação diminuiu ao longo do semestre?	45,6%	53,8%	0,6%
Tenho dificuldade em compreender exatamente o que se pretende perguntar nos exercícios e questões das provas.	49,4%	50,0%	0,6%
Sinto-me desmotivado porque não tenho um feedback da correção dos exercícios.	44,4%	55,6%	
Sou adepto do uso de plataformas digitais de ensino à distância (eLearning), tipo Moodle ou Blackboard.	90,6%	9,4%	
Gostaria de poder usar uma plataforma de apoio à resolução de exercícios fora da aula que desse feedback, mostrasse soluções ou guiasse a resolução.	92,5%	5,6%	1,9%

Tabela 2.1: Resultados percentuais de algumas perguntas feitas a 160 alunos

É possível perceber-se que perto de metade dos alunos assumem que desmotivam à medida que o semestre avança. Idêntica percentagem diz não conseguir compreender facilmente aquilo que lhes é perguntado (pedido para fazer). Provavelmente essa é uma das causas do insucesso e da consequente desmotivação. Também uma percentagem elevada (cerca de 45%) considera que desmotiva devido à falta de *feedback* por parte do professor em tempo útil. Além destas questões diretas, foi perguntado se os alunos se sentiam intrinsecamente motivados para o estudo da matéria ou se precisavam de incentivos externos, tendo-se apurado que 42% reconhece a necessidade desses incentivos (como mostra a Figura 2.12). Esse é um resultado crucial para justificar a preocupação permanente que o docente tem de ter para encontrar novas e mais eficazes formas de melhorar corroborando assim o que foi dito no início do capítulo.

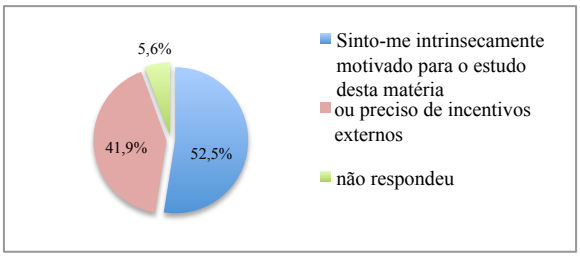


Figura 2.12: Resultados percentuais à pergunta "Sinto-me intrinsecamente motivado para o estudo desta matéria ou preciso de incentivos externos" de 160 alunos

À pergunta “A principal razão pela qual estudo”, 48% responderam que seria para adquirir novos conhecimentos, 33% para tirar boas notas e os restantes dispersaram-se por outras respostas, o que sugere que os alunos muitas vezes precisam das recompensas externas, nomeadamente as notas, o que condiz com o resultado já mostrado na Figura 2.11. Através de outras perguntas, foi ainda possível perceber que os alunos se sentem motivados devido ao “interesse/desafio da matéria” e à “importância dos conteúdos para a vida profissional”. As razões mais escolhidas para a sua desmotivação foram a “dificuldade da matéria” seguido dos dois fatores “modo de funcionamento das aulas” e do “insuficiente acompanhamento por falta de tempo ou por falta de bases”.

2.5.1 Consolidação do estudo efetuado

Como já referido, para consolidar o estudo anterior, foram feitos mais 77 inquéritos a alunos de três cursos diferente em anos escolares diferentes de duas Instituições de Ensino Superior distintas. Os resultados corroboraram as conclusões apresentadas para a primeira amostra, conforme se pode ler na Tabela 2.2 e na Figura 2.13.

	Sim	Não	Não respondeu
Sente que a sua motivação diminuiu ao longo do semestre?	47%	51%	3%
Tenho dificuldade em compreender exatamente o que se pretende perguntar nos exercícios e questões das provas.	53%	45%	1%
Sinto-me desmotivado porque não tenho um feedback da correção dos exercícios.	35%	60%	5%
Sou adepto do uso de plataformas digitais de ensino à distância (eLearning), tipo Moodle ou Blackboard.	60%	39%	1%
Gostaria de poder usar uma plataforma de apoio à resolução de exercícios fora da aula que desse feedback, mostrasse soluções ou guiasse a resolução.	87%	12%	1%

Tabela 2.2: Resultados percentuais de algumas perguntas feitas a 77 alunos

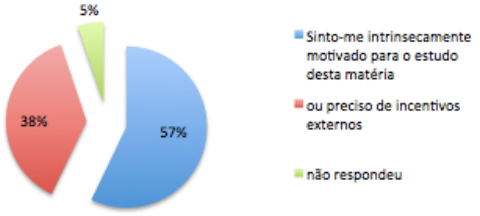


Figura 2.13: Resultados percentuais à pergunta "Sinto-me intrinsecamente motivado para o estudo desta matéria ou preciso de incentivos externos" de 77 alunos

Após a análise convencional de todos os dados recolhidos referentes aos 237 inquiridos, ainda se procedeu a um estudo preliminar com técnicas *Data Mining* com o intuito de perceber que tipo de *regras de associação* se poderiam inferir entre os diferentes fatores e a motivação.

Usando então a função 'apriori' do pacote 'arules' do R[7], com os parâmetros *supp*=0.1 e *conf*=0.9, obteve-se um conjunto grande de regras de associação das quais se selecionaram algumas que se listam a seguir:

[R1] {Face.à.dificuldade.das.matérias.da.disciplina..sinto.me.desmotivado.e.desisto,
Preciso.incentivos.externos} =>
{Sente.que.a.sua.motivação.diminuiu.ao.longo.do.semestre}
(support=0.14, confidence=0.91, lift=1.98)

[R2] {Face.à.dificuldade.das.matérias.da.disciplina..sinto.me.desmotivado.e.desisto,
Gostaria.de.poder.usar.uma.plataforma.de.apoio.à.resolução.de.exercícios.fora.da.aula.que.desse.feedback.mostrasse.soluções.ou.guiasse.a.resolução.,Preciso.incentivos.externos} =>
{Sente.que.a.sua.motivação.diminuiu.ao.longo.do.semestre}
(suport=0.13, confidence=0.91, lift=1.97)

[R3] {Atribuiu.motivação.ao.interesse.desafio.da.matéria, Não.sente.desmotivado.por.não.ter..feedback.da.correção.dos.exercícios, Principal.razão.pela.qual.estuda.é.adquirir.novos.conhecimentos, Sente.que.a.sua.motivação.NÃO.diminuiu.ao.longo.do.semestre}=>
{Sente.se.intrinsecamente.motivado.para.o.estudo.desta.matéria}
(support=0.12, confidence=0.93, lift=1.72)

[R4] {Atribuiu.motivação.à.importância.da.matéria.para.a.vida.profissional,
Não.sente.desmotivado.por.não.ter..feedback.da.correção.dos.exercícios,
Principal.razão.pela.qual.estuda.é.adquirir.novos.conhecimentos,
Sente.se.intrinsecamente.motivado.para.o.estudo.desta.matéria} =>
{Sente.que.a.sua.motivação.NÃO.diminuiu.ao.longo.do.semestre}
(support=0.11, confidence=0.93, lift=1.78)

'arules' é um pacote do sistema R que fornece uma infraestrutura para representar e analisar conjuntos de dados transacionais com o intuito de descobrir padrões tais como regras de associação. Entre outros, esse pacote permite utilizar o algoritmo 'apriori' que é um dos mais usados em *Data Mining*. Este algoritmo funciona em dois passos: num primeiro passo descobre todos os conjuntos frequentes (com suporte pelo menos igual ao suporte mínimo) e

[7] R é uma linguagem e um ambiente para Computação Estatística e gráfica: https://www.r-project.org

num segundo passo extrai desses conjuntos frequentes as regras com confiança suficiente. Estes algoritmos produzem facilmente um enorme número de regras, pelo que se utilizam, para além das medidas de confiança e de suporte, outras medidas de interesse das regras, tais como o *lift*.

Recorde-se que uma regra de associação tem a forma A=>B em que A e B são conjuntos frequentes de itens e expressa uma tendência de se observar B sempre que se observa A. A medida de confiança de uma regra reflete a probabilidade de se observar B numa transação do conjunto de dados sabendo-se que também se observa A na mesma transação. A confiança é calculada empiricamente a partir do próprio conjunto de dados e estima a probabilidade a posteriori de B. Outra medida importante que caracteriza uma regra de associação A=>B é o suporte que se define como a proporção de transações que contêm simultaneamente todos os itens em A e em B. O *lift* é o quociente entre a probabilidade a posteriori do consequente da regra e a sua probabilidade a priori. Ou seja, lift(A=>B) = confiança(A=>B)/suporte(B). Quando o valor de *lift* é próximo de 1, isso significa que A não traz informação relativamente a B e a regra não é interessante. Mesmo que tenha um suporte ou uma confiança elevada (Agrawal et al., 1994; Bayardo et al., 1999; Han et al., 2004).

Mesmo sendo um estudo preliminar, os resultados apresentados corroboram o que foi anteriormente argumentado.

2.6 Sumário

É a motivação que permite a uma pessoa realizar eficazmente determinada atividade, logo é fundamental conhecer o que motiva as pessoas.

Neste documento, após caracterizar a motivação do Ser Humano (em geral, no trabalho e no ensino) e a sua relação com os vários níveis de necessidades que este sente, estudou-se e discutiu-se métodos para atuar como fatores externos estimulando a motivação extrínseca e a autoconfiança dos alunos que facilmente se desinteressam quando enfrentam dificuldades, problema este que é frequente e devastador no ensino da programação.

Os alunos devem ser capazes de adquirir gradualmente confiança através de uma sequência de tarefas cada vez mais difíceis.

O aumento da motivação, via um maior envolvimento dos alunos, pode ser conseguido recorrendo à gamificação, forma de ensinar e aprender através da aplicação dos elementos que se usam na construção dos jogos. O estudo aqui realizado é importante para compreender como este processo de gamificação poderá ser implementado, possibilitando o desenvolvimento de ferramentas adequadas e criando linhas orientadoras que possam ajudar e guiar aqueles que pretendam incluir a gamificação nas suas aulas. Para isso, uma das maneiras de tentar resolver este problema será a utilização de plataformas de apoio ao ensino, com as quais os alunos conseguem acompanhar a evolução das aulas, com o respetivo *feedback*, em tempo útil (na Tabela 1, 92% dos alunos disseram que gostariam de usar uma plataforma de apoio ao seu estudo). A atividade gamificada deve ser planeada tendo em consideração informações como: interesses, necessidades, objetivos, etc. e com a possibilidade de ser melhorada (mudanças no contexto, pelas necessidades do jogador ou para criar novas formas de envolver os jogadores).

Neste capítulo começou-se por analisar as principais dificuldades que se levantam aos alunos quando colocados perante a necessidade de aprender a programar, o que leva a um estudo aprofundado da motivação conforme se acaba de sintetizar.

Neste contexto, e considerando os factos acima referidos, é importante discutir, a estratégia de animação e suas ferramentas, e a importância do *feedback* no processo de ensino-aprendizagem. Nesta perspectiva, é essencial uma análise do impacto das ferramentas de avaliação automática de programas que podem ser integradas no processo de ensino para fornecer o dito *feedback* automático. Estes dois temas serão analisados nos capítulos seguintes.

3. ANIMAÇÃO DE PROGRAMAS: SISTEMAS

Muitas vezes o professor tem a necessidade de usar representações visuais para ajudar os alunos a entenderem os algoritmos, e se for o caso, o comportamento de um programa. Mas para ilustrar o comportamento é preciso mais que uma imagem e a transição entre elas pode igualmente ajudar a compreender o processo. Por isso a animação de algoritmos é importante. A animação pode ser composta por um conjunto de imagens que são apresentadas sequencialmente a um determinado ritmo, de forma mais ou menos interativa. Estas constatações, que imanam da rotina de qualquer docente de programação, estão na base do interesse devotado desde há algumas décadas aos sistemas de visualização e animação de programas que serão discutidos ao longo deste capítulo.

Assim sendo, observa-se um crescente interesse e dedicação, por parte de educadores e informáticos, na construção de sistemas computacionais com o intuito de melhorar a aprendizagem, com base na animação/visualização e simulação. A grande motivação é a de apelar ao potencial do sistema visual humano. Assim, uma vez que os programas computacionais podem ser pouco claros quando apresentados num formato textual, é esperado que o formato gráfico animado contribua para uma melhor compreensão (Pereira,

2002). A visualização pode proporcionar uma metáfora visual para a compreensão de conceitos complicados descobrindo assim, a dinâmica de processos importantes que geralmente estão longe de serem compreendidos pelos alunos. A Visualização tem sido utilizada para melhorar o ensino em diferentes áreas como é o exemplo da química, física e mecânica (Korhonen, 2003). Ao longo deste trabalho de doutoramento, foram estudadas e classificadas várias aplicações de animação de modo a abranger um leque tão vasto quanto possível. Porém, no âmbito desta tese, o tipo de aplicação que se julga ser de maior interesse corresponde ao conjunto de sistemas cuja animação vai sendo progressivamente despoletada como resposta a ações do utilizador, tornando a interação mais realista. Estas aplicações permitem, por exemplo, introduzir valores quando surgem instruções de entrada, ou escolher parâmetros para invocar uma função. Este tipo de ferramenta pode ser usada para comparar diferentes algoritmos para resolução do mesmo problema.

Muitos estudos têm sido realizados para identificar as regras que se devem levar em conta para projetar/criar visualizações e animações de algoritmos que sejam de facto eficazes para o ensino. Ari Korhonen (Korhonen, 2003) explica que a questão fundamental é, no entanto, como se devem aplicar estes artefactos com a finalidade de ajudar os alunos a lidar com conceitos complexos.

Segundo este autor, do ponto de vista pedagógico, uma ferramenta simples para a visualização da execução de um algoritmo pode não ser suficiente. Será sempre importante mostrar a lógica e comportamento de um algoritmo, sendo menos relevante mostrar os seus detalhes de implementação. A ideia principal será que o sistema faça todo o trabalho de cálculo e determinação do fluxo de execução e o aluno só observe o seu comportamento. Adicionalmente o ambiente deve obter e fornecer *feedback* sobre o desempenho do aluno (Korhonen, 2003).

O executável do programa, por si só, em nada ajuda a entender o funcionamento do programa (Pereira, 2002). Por outro lado, a execução em tempo real de um programa é demasiadamente rápida para haver percepção do fluxo de controlo e até da evolução dos valores das variáveis. Tal análise só é possível com a ajuda de um *debugger*, mas para isso o compilador tem de ser prevenido, de modo a gerar código adicional apropriado. Face ao exposto, parece ser uma boa solução recorrer a simulações do comportamento do programa

num ambiente que permita estudar o seu desempenho. Para seguir esta abordagem, o cerne da questão consiste em criar uma poderosa representação interna que permita registar por completo a semântica do programa fonte e que seja propícia a servir de base às visualizações de cada estado e à simulação da execução de cada instrução.

O uso de interfaces gráficas que permitam estabelecer entre o utilizador e o computador uma comunicação não limitada à forma textual tem sido uma preocupação pronunciada por vários autores.

Segundo Maria João Pereira (Pereira, 2002), a animação de um algoritmo é um tipo de visualização dinâmica das principais abstrações expressas pelo mesmo---é pois uma forma natural de expressar comportamentos. A sua importância reside na habilidade de retratar a essência da lógica do algoritmo; sendo uma animação uma conjunção de movimentos de objetos gráficos que tem por objetivo representar comportamentos, mudanças de valores das variáveis, etc.

Assim e à laia de síntese pode dizer-se que uma visualização corresponde a um estado do programa e que uma animação é composta por um conjunto de visualizações. A visualização/animação de *software* tem como objetivo tornar os algoritmos e os programas mais fáceis de entender, recorrendo a representações icónicas para mostrar as abstrações que são usadas para construir um programa. A animação de algoritmos consiste numa visualização dinâmica dessas abstrações de modo a mostrar a sua evolução ao longo do tempo (Pereira, 2002)

3.1 Sistemas de Animação

Nesta secção, serão descritos alguns dos sistemas de animação de programas, mais conhecidos, criados com o intuito de serem ferramentas de apoio ao ensino da programação.

3.1.1 Sistema BALSA

BALSA-Brown Algorithm Simulator and Animator: foi o primeiro sistema textual de animação de algoritmos. Surgiu em 1981 e tornou-se um modelo, tendo servido como protótipo para muitos animadores desenvolvidos mais tarde.

Segundo Connor Hughes e Jim Buckley (Hughes et al., 2004), o mais conhecido e pioneiro sistema de animação de algoritmos é o seu descendente Balsa-II.

O sistema foi escrito em C com o objetivo de animar programas em PASCAL.

Os utilizadores podiam controlar as propriedades de exibição do sistema e, assim, serem capazes de criar, apagar, redimensionar, mover, fazer zoom. Também eram capazes de interagir com o sistema, podendo iniciar, parar e até executar o algoritmo no sentido inverso. A versão original do sistema ainda apresentava as animações a preto e branco (Saraiya, 2002).

3.1.2 Sistema TANGO

TANGO: Sistema de animação textual desenvolvido por Stasko (Stasko, 1990) contribuindo com um modelo de animação com semântica precisa chamada *Path/Transition*, cuja principal função é a criação de movimento contínuo a partir de uma imagem. Isto é conseguido visualizando todos os tipos de animação como uma imagem em movimento ao longo de um trajeto de mudanças incrementais.

O algoritmo de animação em Tango é baseado no do sistema Balsa seguindo o conceito de identificar eventos interessantes no programa.

A versão original do Tango evoluiu para XTANGO, que tem características adicionais, uma das mais importantes será a sua capacidade de animar algoritmos simultâneos (Hughes et al., 2004).

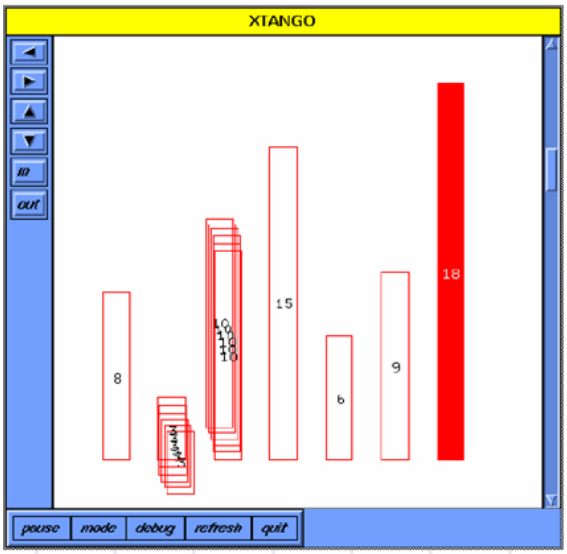


Figura 3.1: Algoritmo de Ordenação animado pelo Sistema XTango

Na Figura 3.1, é mostrada a janela de Xtango, apresentando uma animação de um algoritmo de ordenação. Nesta imagem podemos observar as trocas de posições nos momentos em que os elementos são ordenados (Costacurta, 2008).

3.1.3 Sistema Jeliot

Jeliot[8]: É um sucessor do sistema Eliot, desenvolvido originalmente para animar estruturas de dados de programas escritos em linguagem C, reescrito para animar programas Java. Chamadas a *métodos*, *variáveis* e *operações* são exibidas através do recurso a representações gráficas elucidativas à medida que a animação flui, permitindo ao estudante seguir passo a passo a execução do programa. Este sistema destaca-se pela facilidade de instalação e uso e pela qualidade da visualização apresentada.

A interface deste ambiente encontra-se ilustrada na Figura 3.2 e na Figura 3.3, que mostram momentos da animação de um programa Java que faz o cálculo da média de N notas de um aluno (o respectivo código fonte é mostrado na janela da esquerda, superior).

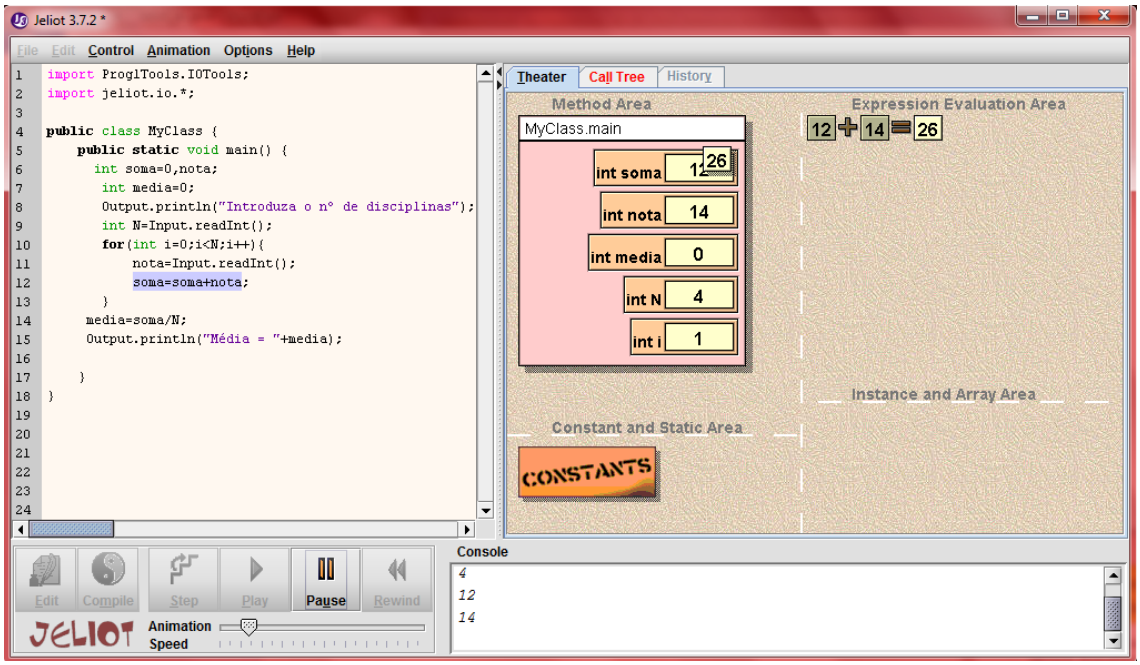


Figura 3.2: Interface do Jeliot - Exemplo do cálculo do somatório de duas notas

[8] https://cs.joensuu.fi/jeliot/

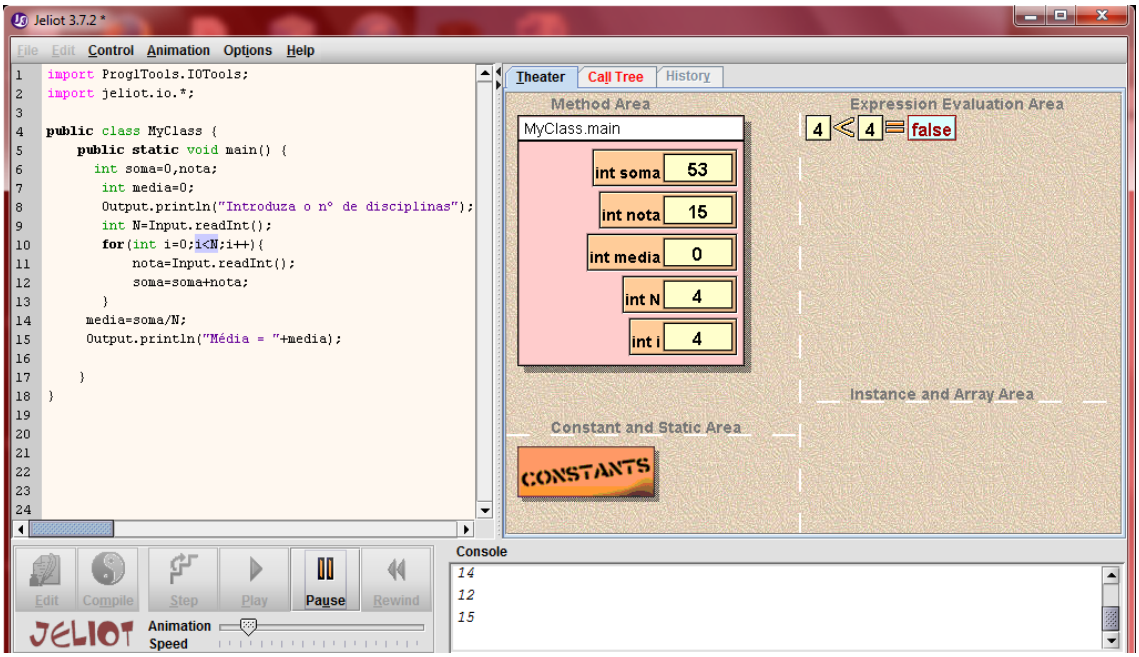


Figura 3.3: Interface do Jeliot - Exemplo da verificação da condição do ciclo de repetição

O *software* é baseado na metáfora teatral, onde o programa a ser apresentado é o texto da peça e os atores são as entidades gráficas que representam as variáveis do programa, sendo possível controlar a aparência e velocidade de cada apresentação.

A janela é dividida em duas grandes partes, uma à esquerda onde se apresenta o código fonte em Java e uma outra à direita, o chamado "teatro", onde é mostrado o que acontece no estado do sistema à medida que o código é executado. Essa janela está subdividida em várias áreas para exibir coisas distintas. Uma dessas áreas contém uma tabela de variáveis onde se pode verificar a alteração de valores à medida que o algoritmo flui; outra mostra as constantes do programa e os valores associados; outra mostra as expressões que estão a ser calculadas e o respectivo resultado obtido, resultado esse que ou é guardado numa variável, escrito na janela de output, ou usado para tomar uma decisão sobre a próxima operação a executar (todos estes movimentos são mostrados através de icons próprios durante a animação).

Na Figura 3.3 mostra-se o cálculo da condição que controla a estrutra de controlo cíclica *for.* (Silva et al., 2009).

3.1.4 Sistema Alma

Alma[9]: Sistema para animar algoritmos independentemente da linguagem, fornece uma ajuda aos programadores e aos professores de Informática, permitindo-lhes validar os algoritmos que desenharam para resolver determinado problema, bem como a própria implementação na linguagem de programação que escolheram (Pereira et al., 1999 ; Cruz et al., 2008)

O sistema Alma surge assim como um instrumento de apoio ao ensino da programação e como instrumento da didática da matemática, entre outras diversas aplicações como, por exemplo, na área da Compreensão de Programas (importante e complexa atividade ligada à manutenção de *software*).

O sistema Alma é adaptável a diferentes linguagens de programação. Cada texto de entrada a ser analisado e animado terá de ser um programa escrito numa linguagem de programação à qual corresponde uma gramática que deverá ser conhecida pelo sistema. O sistema, a partir do momento que consiga reconhecer as frases da linguagem definida por essa gramática, consegue construir uma representação interna de cada texto disponibilizando as informações necessárias sobre o conteúdo dos programas para proceder à sua animação. Esta parte de adaptação do Alma a diferentes situações, embora não seja tarefa a ser executada pelo utilizador final, é um trabalho que pode ser realizado sistemática e facilmente pelo implementador (se este for especialista em desenvolver processadores de linguagens).

A finalidade deste sistema é analisar um programa e extrair a informação considerada importante que, depois de visualizada de uma forma clara e sugestiva, permite entender melhor o significado do programa. Se o texto de entrada for um programa escrito numa linguagem imperativa, a informação a extrair terá que permitir compreender o fluxo de controlo e o comportamento das estruturas de dados. A forma como essa informação irá surgir (ser mostrada) no final deverá depender das escolhas efetuadas pelo utilizador. Por exemplo, pode-se optar por ver só o conteúdo das variáveis, ou só o fluxo das instruções, ou ambos, poder-se-á ver o valor instantâneo da variável, mas também a sua evolução representando os valores no eixo do tempo (Cruz et al., 2009).

[9] http://www4.di.uminho.pt/~gepl/ALMA/

Alma2 (Oliveira et al., 2009) é um sistema, derivado do Alma, que visa proporcionar uma nova forma de compreender os programas escritos em uma Domain Specific Language (DSL). Alma2 fornece ligações entre os domínios do problema e programa. Usa o conhecimento desse domínio específico para criar animações e visualizações dos objetos a nível do domínio do problema, apresentando-as sempre sincronizadas com as visualizações no nível do domínio do programa[10] .

O écran principal de Alma2 é composto por 4 partes importantes, como representado na Figura 3.4:

- Tabela de identificadores (coluna da esquerda, janela superior) Representa o estado interno do programa que está a ser interpretado. Tem a função de manter as variáveis do programa e os seus valores.

- Código fonte (coluna da esquerda, janela inferior) Um campo de texto (não editável) com o código fonte do programa em análise. A linha que está a ser interpretada em cada momento é realçada com uma cor diferente.

- Árvore de Interpretação (coluna da direita, janela superior) É a representação semântica estática/dinâmica do programa de entrada. Esta árvore também representa o fluxo do programa, e usa três cores para indicar que o nó ainda não foi interpretado, o nó que está sob interpretação e o nó que foi interpretado, respetivamente, cinza, vermelho e verde.

- Painel de Animação (coluna da direita, janela inferior) É a janela onde as cenas com os atores usados nas animações são desenhadas. Nessa janela o sistema Alma2 representa os efeitos da execução/interpretação do programa sobre os conceitos do domínio do problema e esta é que é de facto a grande novidade e o principal contributo desta ferramenta.

[10] http://www3.di.uminho.pt/~gepl/Alma2/index.php

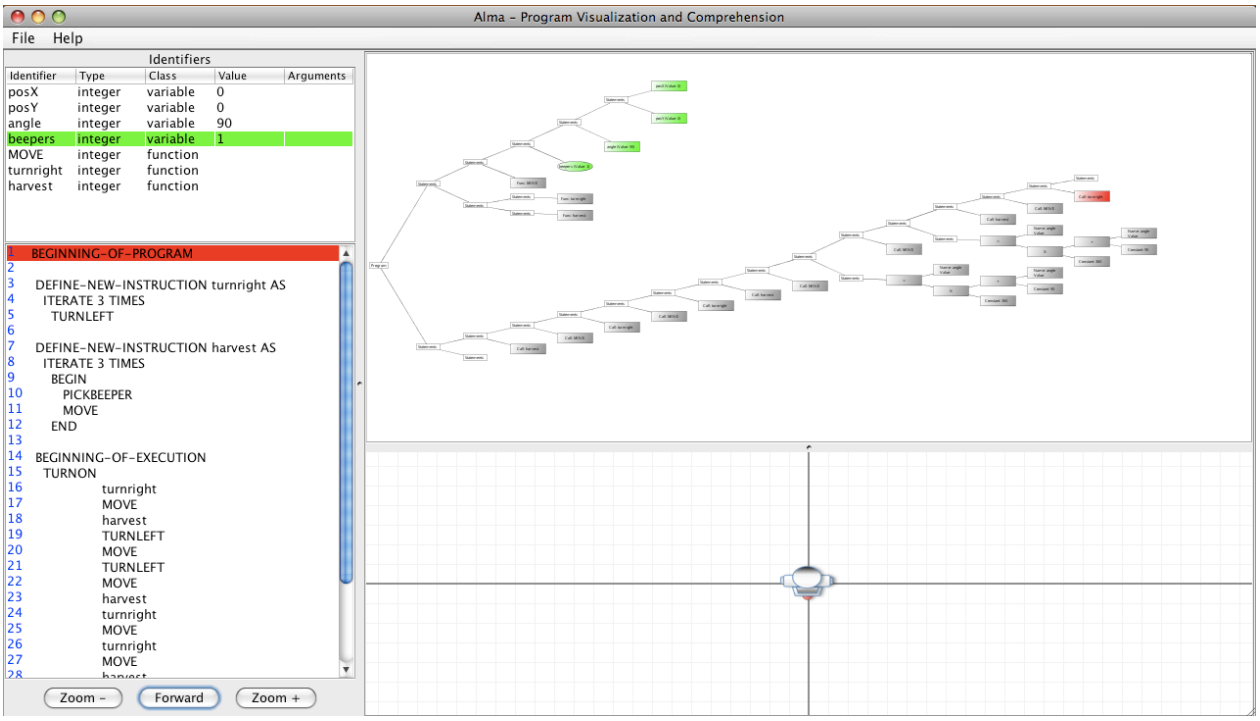


Figura 3.4: Interface do Sistema Alma

3.1.5 Sistema AMBAP

AMBAP[11]--Ambiente de Aprendizagem de Programação: ferramenta para simular a execução de algoritmos. Os algoritmos, escritos através de um fluxograma ou em português estruturado, são analisados por um interpretador que simula a sua execução permitindo ao aluno visualiza-la passo a passo. Assim sendo, este ambiente apoia o aluno durante o processo de resolução de um problema, possibilitando que este compreenda e valide o respetivo algoritmo antes de passar para a codificação do programa, libertando-o da complexidade sintática e semântica das linguagens de programação concretas (Xavier et al., 2004)

Apesar do sistema não ter sofrido evoluções significativas desde 2001, manteve-se a funcionar durante muitos anos, sendo aqui referido por essa razão e especialmente por ter sido uns dos primeiros animadores de algoritmos em Português Estruturado. Contudo outros sistemas de animação de algoritmos mais recentes serão apresentados em secções seguintes, pelo que entendemos não se justificar entrar em mais detalhes sobre o AMBAP.

[11] https://sites.google.com/site/ldsicufal/softwares/projeto-ambap

3.1.6 Sistema SICAS

SICAS--Sistema Interativo para Construção de Algoritmos e sua Simulação(Gomes et al., 2004; Mendes et al., 2000): Ambiente que de forma dinâmica, interativa e apelativa permite a construção e simulação de algoritmos. Fornece um ambiente que pode ajudar os alunos não apenas a compreender o funcionamento de um algoritmo, mas que sobretudo lhes permite escrever, executar, testar, alterar e corrigir os seus próprios algoritmos. Desta forma, o SICAS apresenta não uma abordagem descritiva dos assuntos a ensinar/aprender, em que o aluno recebe a informação passivamente, mas antes oferece um ambiente que permite que os alunos desenvolvam as suas resoluções com base na experimentação e na prática. Esse ambiente permite ao aluno a simulação da execução, observação, e análise visual da forma como um dado algoritmo funciona ou se comporta mediante certas circunstâncias, possibilitando-lhe compreender a semântica dinâmica do programa, detetar eventuais erros, corrigi-los e através de tudo isso aprender ao seu próprio ritmo. Este processo interativo e visual de detecção e correção de erros (via animação da execução) é fundamental para o desenvolvimento nos alunos de competências de programação.

A escrita da solução do problema é realizada, no SICAS, através de fluxogramas. Para a escrita de expressões, o SICAS não segue uma sintaxe rígida de uma determinada linguagem de programação, porém segue uma notação próxima do C e do JAVA. No entanto, há um esforço constante para minimizar os detalhes sintáticos, de forma a que o aluno se possa concentrar plenamente na sua tarefa principal. O SICAS possui também um conjunto de funções intrínsecas, pré-definidas que podem ser utilizadas na construção de expressões sem que haja a preocupação da sua declaração. As funções existentes no SICAS são as mais frequentemente utilizadas na generalidade das linguagens de programação, que proporcionam a resolução de grande número de problemas típicos da fase inicial de aprendizagem deste tipo de matéria. O SICAS permite também que o aluno defina as suas próprias funções.

Como se disse, o SICAS é um ambiente que possibilita, essencialmente, dois tipos de cenários: edição/resolução de problemas e execução/simulação de resoluções previamente construídas. No primeiro cenário, o utilizador pode construir algoritmos através de representações visuais – fluxogramas. No segundo cenário, o utilizador pode simular a execução das resoluções construídas, analisando-as com o detalhe e ritmo desejado.

Neste contexto, foram consideradas as seguintes operações para a construção de fluxogramas: Atribuição (tem como finalidade atribuir o valor de uma determinada expressão a uma variável), Entrada/Saída (possibilita ler (Entrada) valores para variáveis e escrever (Saída) expressões), Repetição (permite realizar a execução de uma ação um determinado número de vezes) e Seleção (tem como propósito escolher em alternativa o conjunto de ações a ser executado). A Figura 3.5 ilustra a construção de um algoritmo no SICAS.

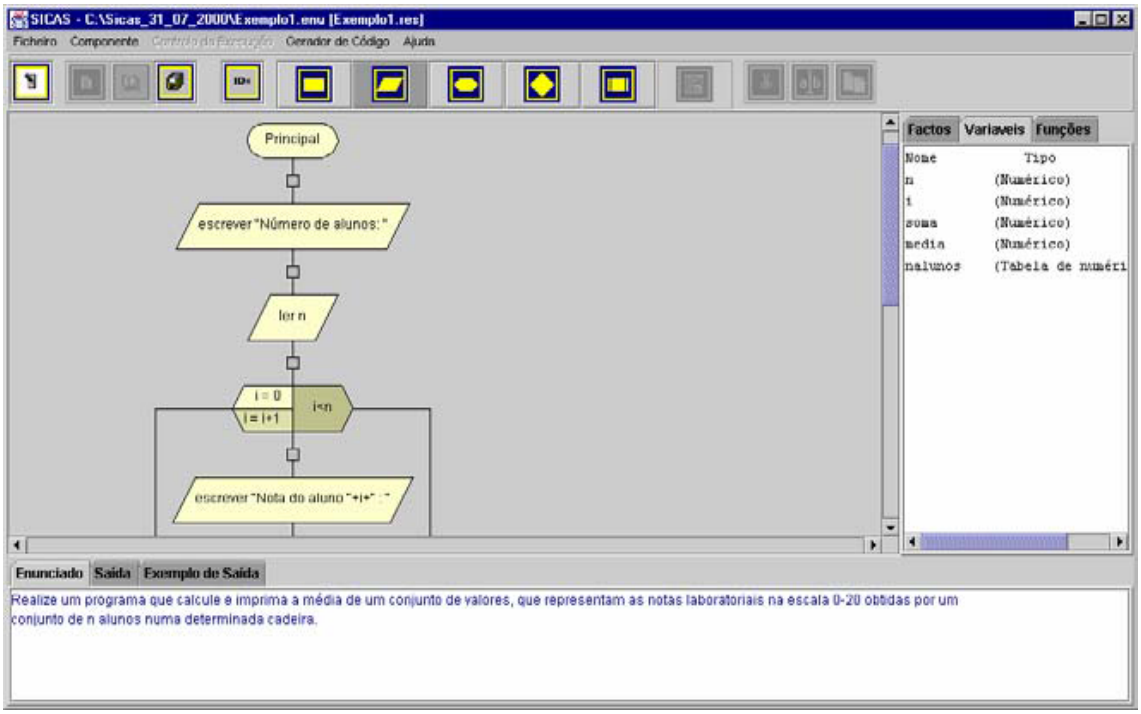


Figura 3.5: Interface SICAS (Modo aluno)

A forma de introdução destes elementos é muito simples, bastando para tal selecionar o tipo de elemento a introduzir, preencher, numa caixa de diálogo que surge automaticamente, os campos necessários para a completa definição de cada estrutura e selecionar o ponto de inserção pretendido.

Como se vê na Figura 3.5, este ambiente é constituído essencialmente por três áreas: área de construção/edição do algoritmo, área de dados e área de geração de resultados e descrição do problema.

Área de construção/edição do algoritmo: esta é a área destinada à conceção de resoluções de problemas. O SICAS possibilita, desta forma, a existência de dois tipos de cenários: resolução de problemas e execução/simulação de resoluções previamente construídas. Para o primeiro

caso o aluno não terá forçosamente de construir uma resolução de raiz, podendo optar por editar/alterar uma previamente construída por si ou uma das resoluções que o professor tenha disponibilizado para esse problema. Para a criação/edição de uma resolução o aluno é direcionado para um ambiente semelhante, que lhe dá a possibilidade de construir, sob a forma de fluxograma, uma sequência de operações (ações) que, quando executadas, produzirão a solução do problema. A construção de um fluxograma é realizada recorrendo a simbologia gráfica standard que representa as principais estruturas necessárias à construção de um algoritmo, representadas na parte superior desta área. A sequência é representada por uma linha interligando, dois a dois, os componentes constituintes do programa. A criação de uma estrutura envolve a escolha do seu tipo através da seleção do elemento adequado, a sua introdução no local desejado no fluxograma e o preenchimento completo dos campos necessários à sua total definição. É de salientar que a definição e localização de qualquer estrutura jamais terá um carácter definitivo, podendo o aluno em qualquer momento apagar, modificar ou copiar um dos componentes bem como proceder à alteração da sua definição.

Área de geração de resultados e descrição do problema: esta área é constituída pelas subsecções Saída, Enunciado e Solução. A subsecção Saída, como o próprio nome sugere, contém a saída gerada pela execução do programa e também o eventual código produzido, sob a forma de pseudo código, Linguagem C ou Linguagem JAVA. A subsecção Enunciado é destinada à apresentação do problema a resolver. A subsecção Solução destina-se à indicação, pelo professor, de um conjunto de caracteres representativos da saída esperada para o problema definido, assumindo o conjunto de valores de entrada estabelecidos na secção "Entradas" da "Área de dados auxiliar" (Figura 3.6).

Área de dados auxiliares: esta área possui as subsecções Factos, Variáveis e Entradas. A subsecção Factos permite conhecer, em cada momento, os dados da experiência, estabelecendo para isso a correspondência entre cada variável e o valor que lhe está associado. Este par valor/variável é constantemente atualizado durante a execução da resolução e é de crucial importância para a deteção de eventuais erros lógicos de conceção. A subsecção Variáveis, apesar de estar presente quer quando o aluno edita, quer quando simula algoritmos, tem como objetivo permitir a realização de uma consulta rápida a fim de obter

informação sobre as variáveis declaradas e dos respetivos tipos. A subsecção Entradas, embora não visível no modo aluno, tem como finalidade, em conjunto com a subsecção Solução, permitir que o aluno teste a correção da sua resolução. Assim, esta subsecção destina-se a permitir ao professor fornecer um conjunto de dados de entrada, os quais estabelecem uma correspondência unívoca com as operações de leitura executadas durante a simulação do algoritmo.

Após a construção do fluxograma que descreve o algoritmo delineado para ser a solução do problema, este poderá ser simulado pelo sistema. Para isso, o aluno terá que transitar para outro ambiente - modo de execução, como representado na Figura 3.6. Este ambiente permite um elevado grau de interação e controlo por parte do aluno, possibilitando-lhe realizar as seguintes configurações: escolha da direção da execução que permite que a execução se processe normalmente da primeira instrução para a última ou recuar na execução (permitindo ao aluno voltar a observar uma situação anterior); escolha do tipo de movimento realizado que permite estipular o ritmo de execução, só avançando de uma instrução para a que lhe sucede quando o aluno o autorizar (Passo-a-passo) ou com uma execução contínua mas lenta ou ainda uma execução ininterrupta mas mais acelerada.

Independentemente da direção e movimento estipulados para o decorrer da execução, o aluno poderá em qualquer momento, parar a execução pelo período de tempo que desejar e posteriormente retomá-la. No decurso de qualquer simulação o próximo componente a ser executado é devidamente destacado, de forma a melhor localizar e enquadrar o aluno na ação que está a decorrer. Durante a simulação o aluno poderá acompanhar a evolução da execução da sua solução, conjuntamente com a inspeção das secções que fornecem informação valiosa para essa apreciação, nomeadamente as secções Factos e Saída.

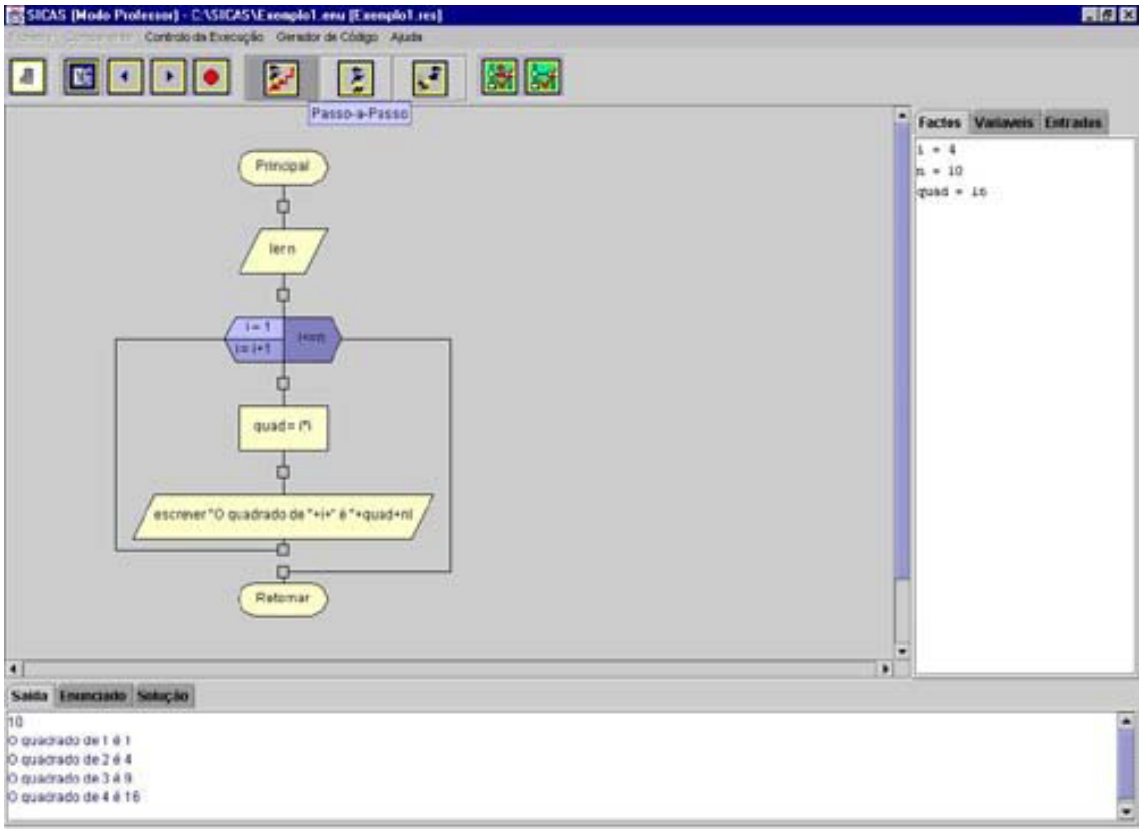


Figura 3.6: Interface SICAS - Modo Execução

Relativamente aos tipos de dados, o SICAS possibilita apenas a utilização de quatro tipos: “Numéricos”, “Cadeia de Caracteres”, “Tabela de numéricos” ou “Tabela de cadeia de caracteres”. A ideia é a de sensibilizar os alunos para os tipos essenciais de dados pois a diversidade de tipos de dados, apesar de ser um aspeto importante das linguagens de programação, constitui muitas vezes, numa fase inicial de aprendizagem da programação, um conjunto de dificuldades acrescidas desviando a concentração dos alunos do importante, a concepção do algoritmo. Para destacar a importância do ato de declaração de variáveis, o SICAS não permite a utilização de variáveis não declaradas.

Neste domínio, existe ainda o sistema H-SICAS (Santos et al., 2010), uma nova versão do SICAS desenvolvido para plataformas móveis como Smart Phones .

3.1.7 Sistema OOP-Anim

OOP-Anim (Esteves et al., 2004; Santos et al., 2010): Este sistema, baseado nas ideias subjacentes ao SICAS, tem a intenção de ajudar os alunos a compreender os conceitos básicos da programação orientada a objetos através da criação e visualização de programas que sigam essa abordagem. Como o SICAS, OOP-Anim permite a simulação do programa, mas neste caso, tem que ser escrito em Java. Assim, os alunos podem entender os efeitos de todas as instruções existentes em termos de definições de classes, instâncias de objetos e referências relacionadas. A sua operação começa com um programa Java, normalmente escrito pelos alunos, e depois disso é possível controlar a execução de código e visualizar o efeito de cada instrução em termos de saída e as instâncias de objetos que serão criados, usados e modificados. É possível observar como as instruções afetam as classes e os objetos desse programa.

Para atingir todos os objetivos, os alunos devem praticar intensivamente o desenvolvimento e *debugging* de programas. Este ambiente pode ajudar, uma vez que utiliza animação para facilitar a compreensão do programa e detecção / correção de erros. Este processo tem um grande potencial educativo, porque os alunos podem aprender ao corrigir os seus próprios erros. Na opinião dos autores, quando os alunos chegam a uma solução a sua experiência e confiança normalmente melhora, facilitando assim a aprendizagem.

De acordo com (Esteves et al., 2003) o écran do OOP-Anim tem quatro áreas essenciais como é mostrado na Figura 3.7. O lado esquerdo da janela mostra o código do programa e realça a linha que está a ser executada. A área abaixo do código do programa exibe a entrada e saída do programa. A metade direita da janela tem duas partes principais, a parte superior e a parte inferior. A parte superior mostra a animação do programa. Esta parte está subdividida em três zonas, uma para uma representação das classes do programa, outra para objetos criados durante a execução e as referências utilizadas pelo programa.

A parte inferior mostra os botões usados para controlar o sistema: abrir um programa para a sua animação; fechar o programa e limpar a área de animação; fechar o ambiente; iniciar a animação de forma contínua: parar a animação; executar passo a passo; voltar a examinar estados de programas anteriores.

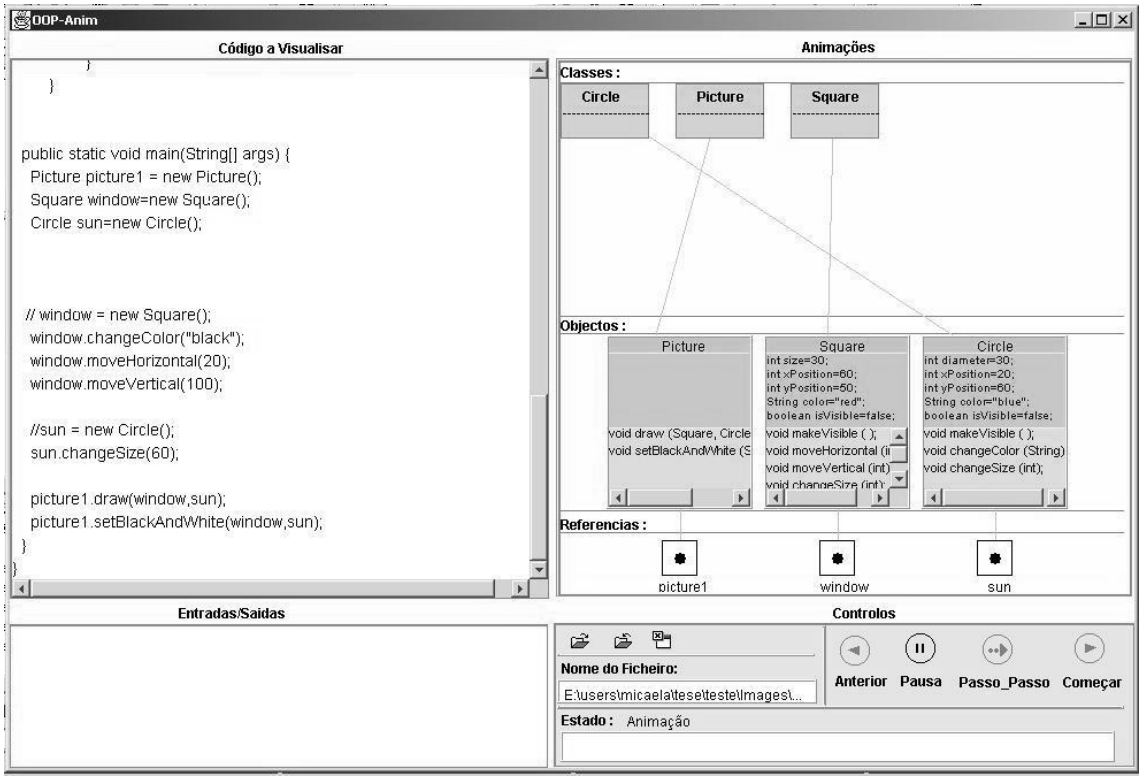


Figura 3.7: Ambiente de OOP-Anim

Por último, existe uma área de estado que mostra o estado do programa (em execução, parado ou à espera de entrada), e uma breve explicação do que está a acontecer em todas as fases de animação.

3.1.8 Sistema VILLE

VILLE [12] (Rajala et al., 2008): Ferramenta de visualização e de animação para apoio a professores e alunos em cursos de ensino da programação. Pode ser usada para criar e editar exemplos de programas desenvolvidos pelos próprios alunos ou pelo professor e observar os eventos durante a sua execução. O seu principal objetivo é apoiar o processo de aprendizagem dos alunos. É possível ao professor adicionar exemplos ao VILLE para, em seguida, o aluno visualizar sua execução, por exemplo, através da web. Um dos aspectos mais importantes de VILLE é a capacidade de visualizar exemplos de programação em diferentes linguagens, incluindo execução paralela de um programa em duas ou mais linguagens e a

[12] http://ville.cs.utu.fi

capacidade de definir novas linguagens. Assim, o aluno, pode descobrir semelhanças nas funcionalidades básicas. É muito mais importante para o programador aprender os conceitos de programação do que se concentrar nas questões sintáticas de um idioma específico.

VILLE contém um conjunto pré-definido de exemplos de programação agrupados em categorias. Assim, podem ser criadas novas categorias e exemplos ou editar os pré-definidos. O sistema fornece informações sobre as variáveis do programa e os valores que vão assumindo durante a execução, sobre o fluxo de execução exibindo as condições e as estruturas de controlo e realçando as chamadas a subprogramas.

A sua eficácia foi avaliada, e com base nos resultados verificou-se que VILLE melhora a aprendizagem dos alunos. A Interface VILLE consiste em cinco janelas separadas: janela principal, componente de animação, criação e edição de exemplos, o editor de sintaxe e editor de perguntas.

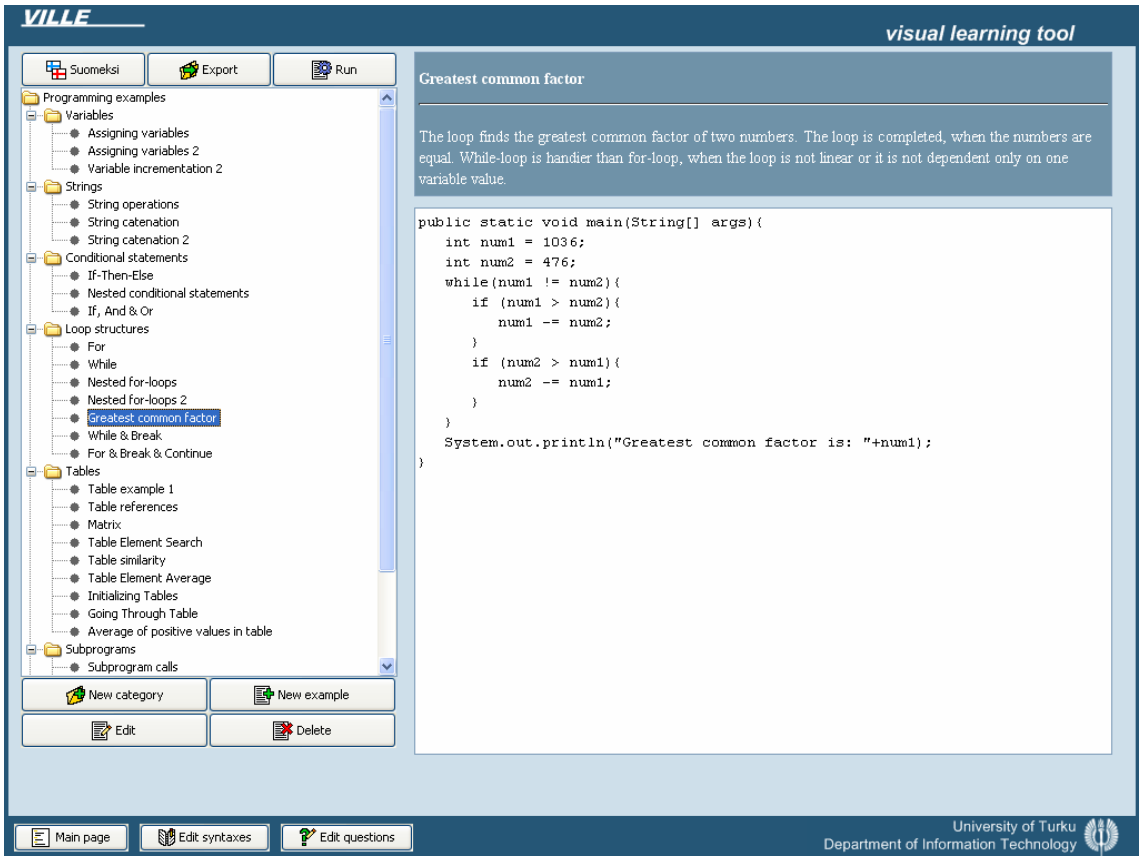


Figura 3.8: Janela principal

Quando a aplicação é inicializada, a janela principal é carregada. Do lado esquerdo da janela (Figura 3.8) encontra-se a árvore de exemplos de programas e botões para controlo da aplicação. Com os botões que se encontram em baixo o utilizador pode modificar os exemplos. Com os botões acima dos exemplos é possível alterar o idioma da aplicação, exportar os exemplos para uma coleção exemplos e executar o exemplo escolhido. O lado direito da janela mostra o código e a descrição do exemplo.

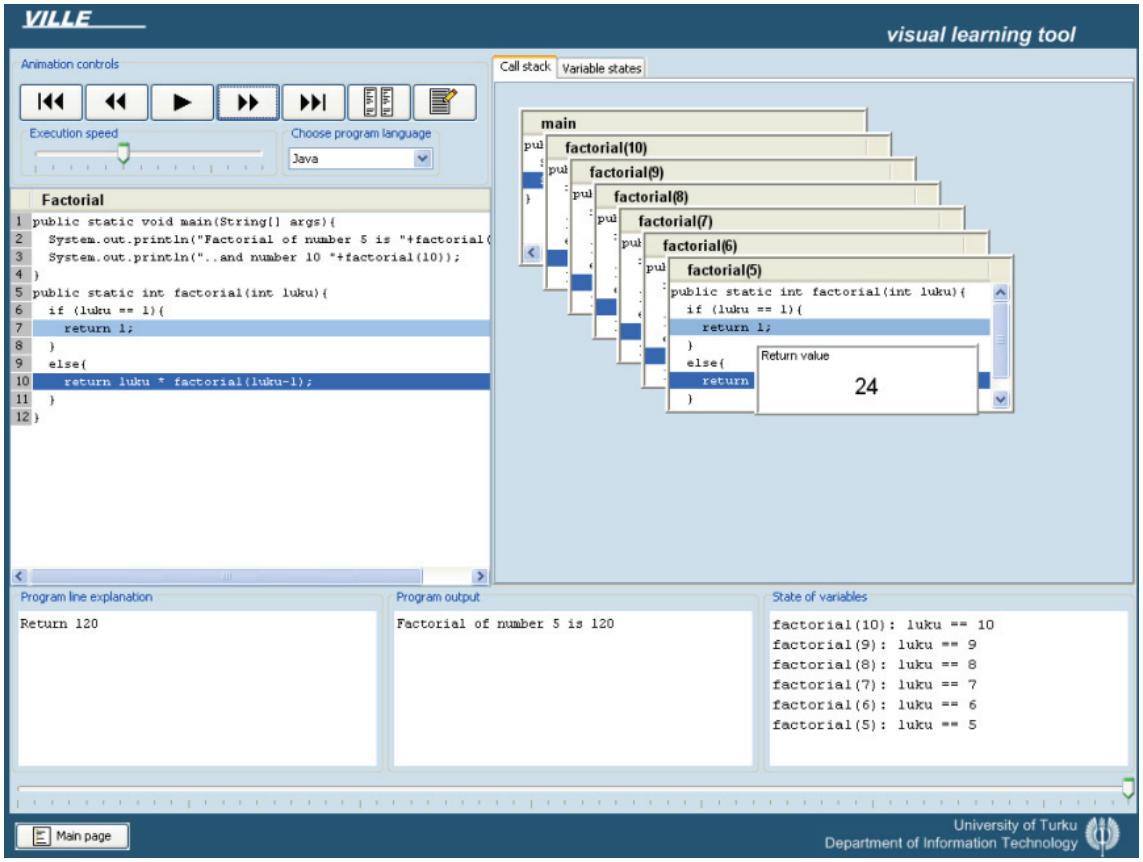


Figura 3.9: Janela da componente da Animação do VILLE

A janela de animação do VILLE, que é apresentado na Figura 3.9, é composta por três áreas. O lado esquerdo contém todos os controlos do ambiente e o código do programa. Esses controlos podem ser utilizados para avançar ou retroceder na visualização. O lado direito exibe a chamada do método, ou seja, cada chamada do método cria uma nova janela e quando a execução é terminada é mostrado o valor a retornar. As janelas na parte inferior da interface, exibem uma explicação sobre a linha atual em que o programa se encontra,

resultados e estados das variáveis. Também é possível visualizar neste ambiente arrays e matrizes com apresentações gráficas (Rajala et al., 2008).

Na janela de criação e edição de exemplos de programação (Figura 3.10) o utilizador pode adicionar código de um programa em Java na área de texto à esquerda. Quando o botão "translate" é pressionado, VILLE cria traduções para pseudo-código, C++ e para todas as linguagens definidas, gerando automaticamente explicações para cada linha do programa.

Por cada novo idioma adicionado ao sistema VILLE é preciso definir o conjunto de regras de tradução que fazem corresponder a cada instrução da linguagem Java as instruções correspondentes na nova linguagem. Durante a animação do programa Java em execução essas regras são usadas para se irem geradas as correspondentes instruções nos idiomas paralelos.

VILLE suporta um subconjunto da sintaxe Java considerado suficiente para o ensino de introdução à programação. Permite lidar com os tipos básicos de variáveis, com as principais funcionalidades da classe String, com instruções condicionais, com estruturas de repetição, com vetores e matrizes, com métodos e funções. Com estes construtores, as funcionalidades básicas de programação podem ser bem ilustradas.

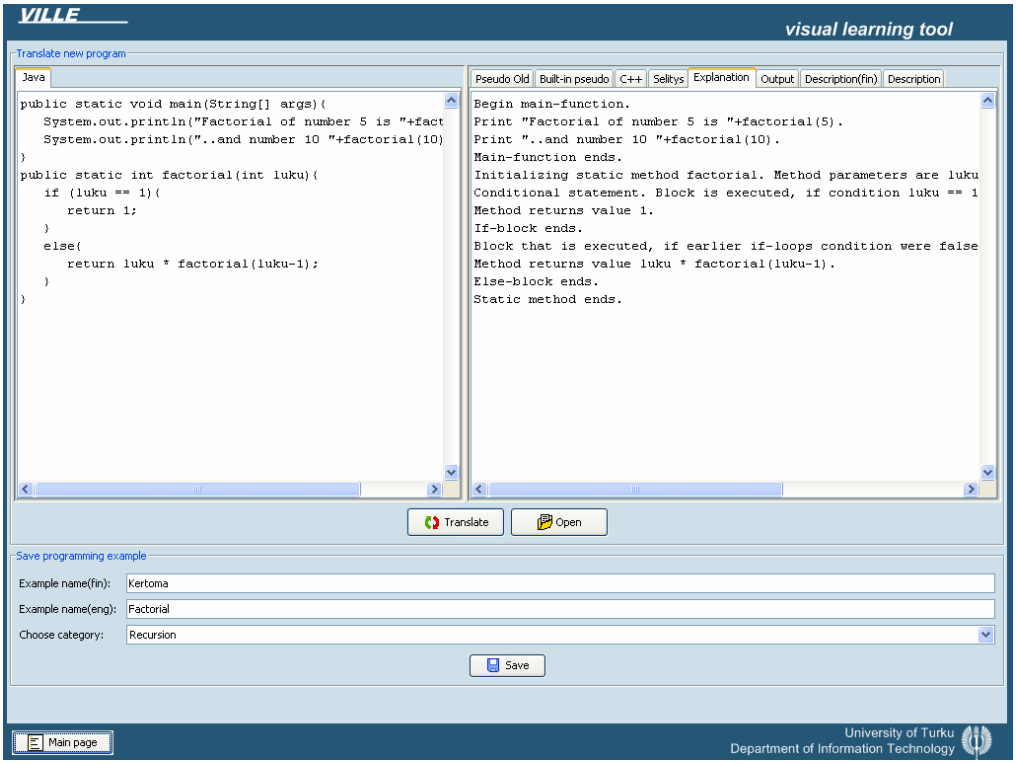


Figura 3.10: Janela da criação e edição de exemplos de programação

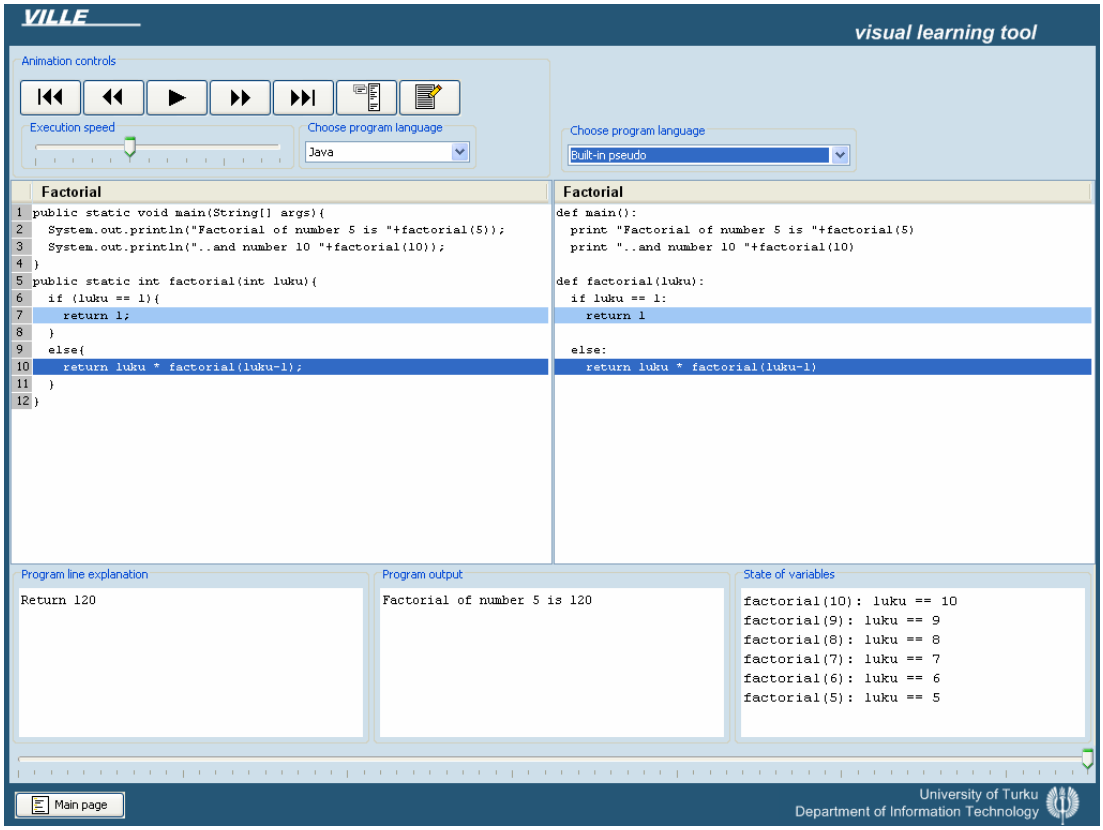


Figura 3.11: Animação de um Programa em 2 linguagens em paralelo (Java e Pseudo-Código)

A Figura 3.11 mostra a execução de um programa, no modo de exibição de visualização em paralelo, em que o código do programa é visto em duas linguagens em simultâneo. Comparando as duas versões do mesmo algoritmo e visualizando a sua execução ao mesmo tempo, o aluno terá a chance de melhor compreender o seu significado, ou seja, a chamada dinâmica.

Com o editor de sintaxe (Figura 3.12), o professor pode, como já foi dito atrás, adicionar novas linguagens de programação ao sistema, definindo a sua sintaxe e as já mencionadas regras de tradução (entre Java e essa nova linguagem). O editor exibe a sintaxe Java nas linhas do lado esquerdo. No lado direito da janela, o utilizador pode selecionar uma sintaxe e modificar ou criar novas sintaxes.

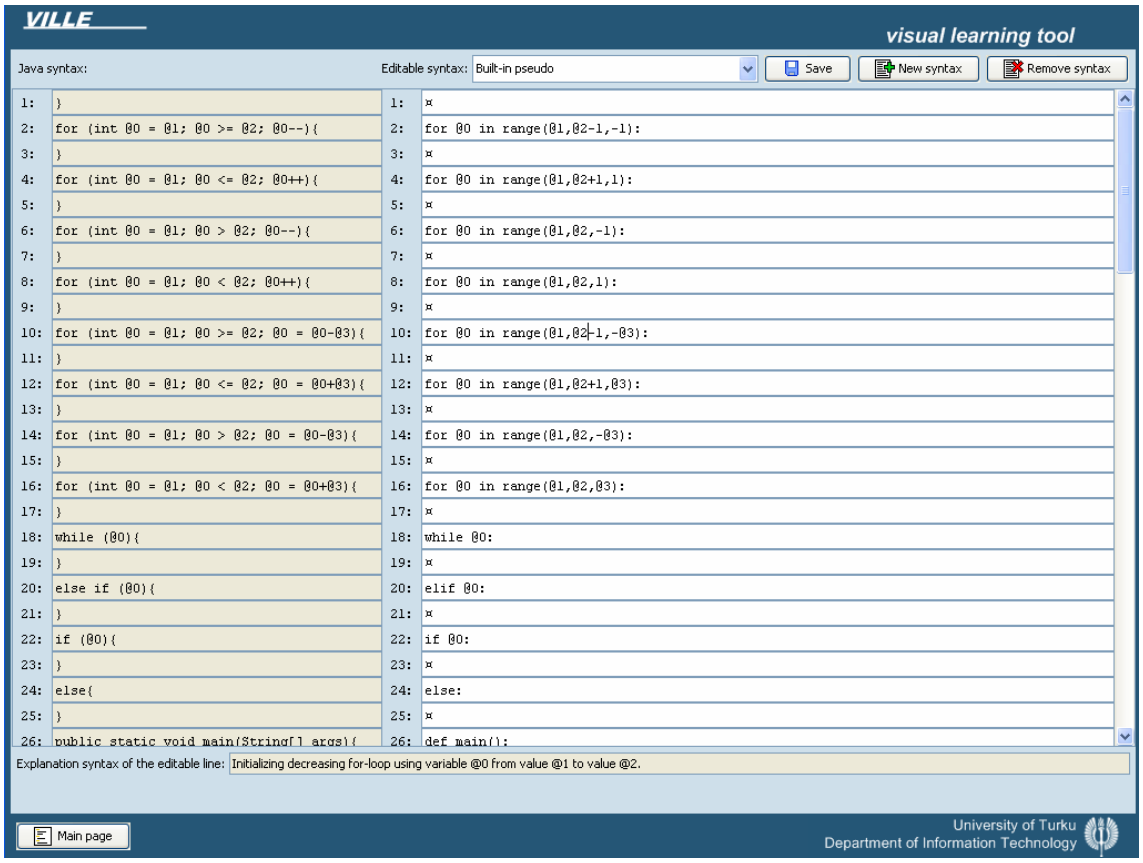


Figura 3.12: Editor de Sintaxe

No editor de Perguntas (Figura 3.13 e Figura 3.14), o utilizador pode criar questões de escolha múltipla e associá-las às linhas de código do programa original de forma a que essa questão de escolha múltipla seja apresentada ao utilizador aquando da simulação no momento em que essa será a próxima instrução a ser executada. Do lado direito da janela, o utilizador pode ler a pergunta, selecionar a resposta e prosseguir. A ausência da possibilidade de retroceder, pode ser frustrante ao executar programas. Com VILLE é possível retroceder ou avançar para qualquer ponto da execução do programa.

Gera também automaticamente uma linha de descrição para cada linha de código do programa executado. A descrição inclui a informação da função, das variáveis e do estado das mesmas. Isso ajuda os alunos a interpretar os acontecimentos nas linhas de código executadas. Todos os exemplos predefinidos, que podem ser visualizados na janela principal, podem também ser publicados na internet (Rajala et al., 2007) .

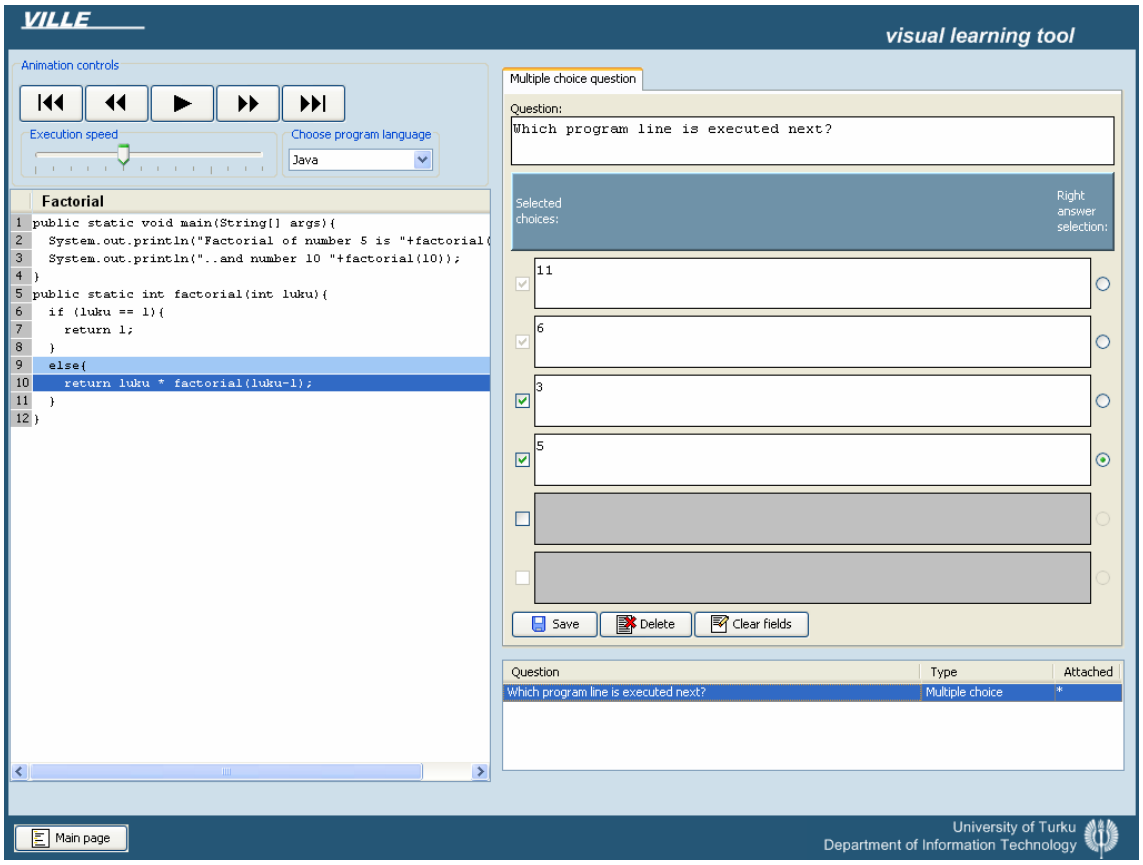


Figura 3.13: Editor de Perguntas

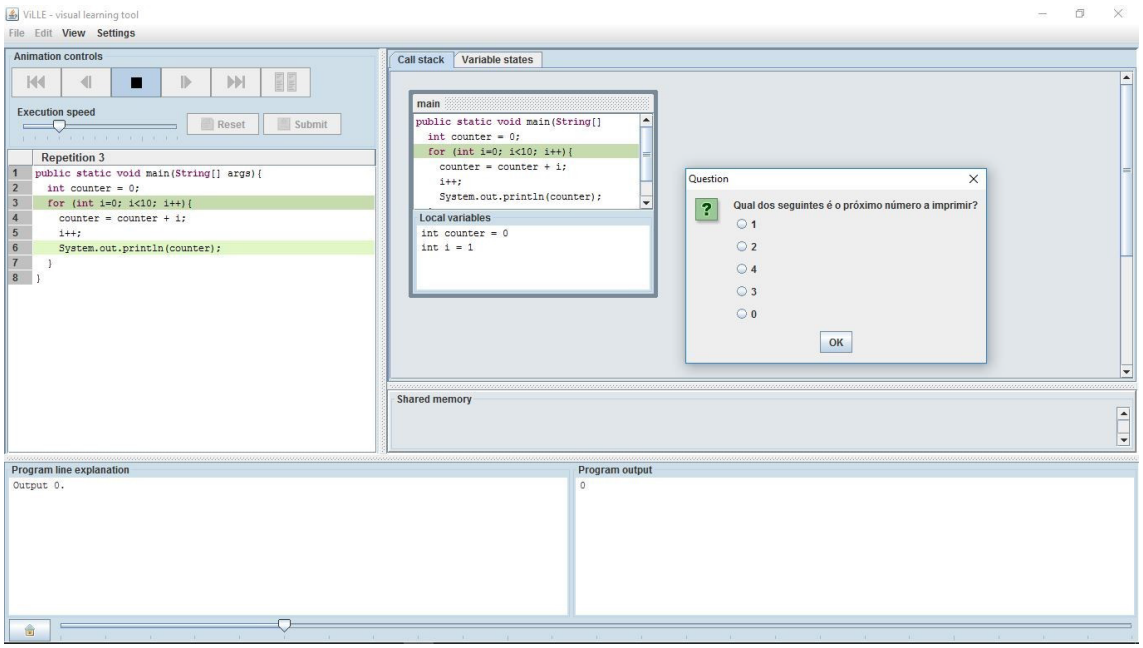


Figura 3.14: Visualização de uma pergunta

Segundo os autores Rajala, M.Laakso, E.Kaila e T. Salakoski (Rajala et al., 2007) aprender a programar é uma tarefa desafiadora, e um dos principais passos para uma melhor

aprendizagem é ir além das questões sintáticas para compreender os conceitos básicos de programação. Além disso, a compreensão do principio de uma linguagem independente deve ajudar na adaptação a novas linguagens de programação e na mudança de um linguagem para outra.

Os autores também desenvolveram um estudo para comparar este sistema, VILLE, com Jeliot3 (ferramenta já discutida numa das secções anteriores). Ambas permitem a execução do código do programa passo a passo mas existem diferenças entre estas duas ferramentas. Em relação à linguagem, Jeliot apenas permite a utilização do Java enquanto que o VILLE permite Java, C++ e pseudo-código, ou outra linguagem definida pelo utilizador. Os controlos são muito semelhantes em ambas as aplicações, mas com VILLE é possível retroceder a execução e avançar para qualquer ponto da execução. Com Jeliot é possível visualizar as alterações do estado das variáveis com simbologia gráfica e a execução de uma instrução é apresentada com mais detalhe do que em VILLE. Ambas as ferramentas destacam a linha de código em execução, mas em contrapartida VILLE também destaca a linha anteriormente executada para ser mais fácil determinar a origem da execução. Jeliot permite pedir ao utilizador os dados de entrada, o que não será possível em VILLE, contudo, com esta ferramenta os alunos podem responder a perguntas durante a execução do programa.

É importante assinalar que o campo de aplicação desta plataforma tem alargado, sendo agora usado no apoio a disciplinas de matemática e programação a nível do ensino básico e secundário, o que levou a aumentar bastante a equipe de desenvolvimento e manutenção/apoio e a evoluir a sua funcionalidade. Recomenda-se a consulta do novo endereço desta plataforma[13], para uma completa percepção do crescimento do VILLE e do seu relevo como ferramenta de suporte ao ensino.

3.1.9 Sistema JIVE

JIVE--Java Interactive Visualization Environment)[14] (Lessa et al., 2011): Ferramenta pedagógica para ajudar a explicar o comportamento dinâmico de programas em Java. Jive pode ajudar os alunos em duas áreas importantes do ensino da programação: na construção mental de modelos de execução orientados a objetos e no *debugging* dos programas. Representa uma

[13] http://villeteam.fi/en

[14] https://www.cse.buffalo.edu/jive/

abordagem no momento de execução, visualização e análise de programas em Java facilitando o entendimento dos programas e o *debugging* com várias visualizações personalizáveis do objeto em análise. Permite a execução via diagramas de sequência, consultas interativas sobre o comportamento e permite ainda uma execução interativa. Esta ferramenta incorpora várias caraterísticas interessantes e inovadoras para este paradigma OO conforme mostrado na Figura 3.15.

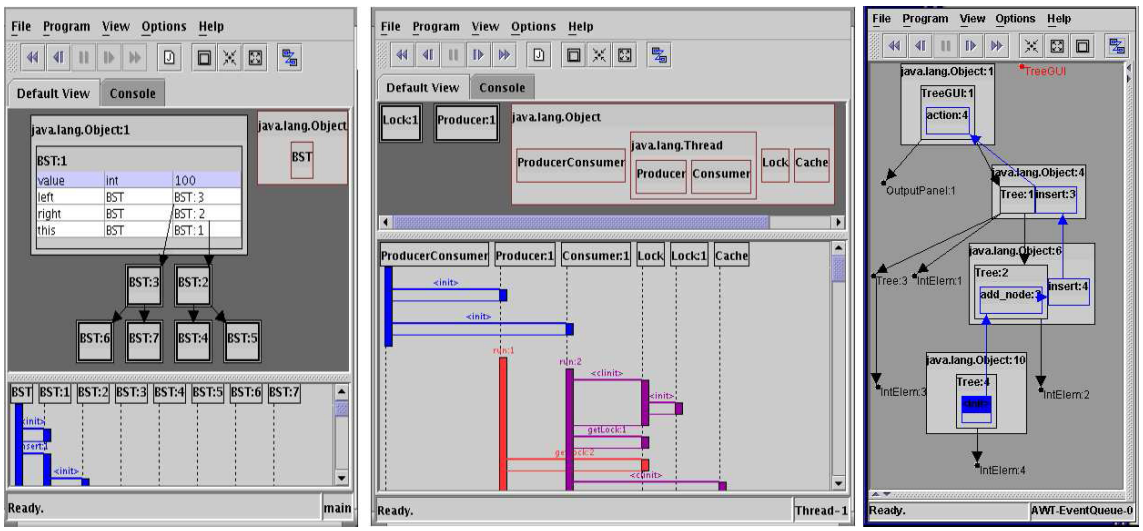


Figura 3.15: Interface do JIVE

A janela da esquerda mostra todos os detalhes sobre um determinado objecto selecionado incluindo a árvore de subobjectos que têm esse nó como raiz. A janela do meio mostra um diagrama que ilustra a interação de vários objetos num programa do tipo *produtor / consumidor*. A imagem mais à direita mostra a execução do método através da interação entre os objetos ativos em diferentes contextos representando todos os objetos inativos minimizados. As imagens da esquerda e da direita mostram como diferentes níveis de detalhe podem ser combinados do ponto de vista do diagrama de objetos.

Jive é capaz de descrever objetos como ambientes utilizando uma nova notação para a exibição de estruturas no momento de execução do objeto, esclarecendo o importante facto de que os objetos são ambientes para execução de programas. Esta notação representa visualmente a relação entre métodos objectos dinâmicos e contextos estáticos.

Jive também torna possível avançar ou retroceder na execução do programa, à semelhança do Jeliot, VILLE e outros sistemas. Esta funcionalidade interativa é importante quando se utiliza Jive para o debugging, uma vez que os erros são necessariamente detetados depois de terem ocorrido.

Oferece ainda suporte para consultas sobre o Estado das variáveis. Compreender o comportamento das variáveis ao longo do tempo é essencial para compreender a execução de um programa. O utilizador deve ser capaz de consultar o sistema: quando as variáveis foram alteradas, quando as variáveis mantiveram certos valores, quando os objetos são criados e quando os métodos são chamados. Os resultados destas consultas devem ser integrados no ambiente de visualização.

Jive usa a Máquina Virtual do Java (JVM), pelo que não são necessários nem um compilador personalizado nem uma nova máquina virtual (Gestwicki et al., 2004)

3.1.10 Sistema VisuAlg

VisuAlg3[15]: é um Editor e Interpretador de Pseudocódigo, ou seja, de uma Linguagem Algorítmica criada para introdução aos conceitos básicos de programação. O Pseudocódigo permite a escrita de algoritmos, do mais simples ao mais complexo, e o VisuAlg oferece aos alunos das disciplinas de Programação a possibilidade de testar os seus conhecimentos executando esses algoritmos num ambiente visual de animação próximo da realidade. A principal ideia do autor (Almeida, 2013) é ajudar a ultrapassar a dificuldade de elaborar os algoritmos no papel (o que para muitos alunos é pouco interessante) e evitar submeter logo à partida os alunos ao rigor e à complexidade de uma linguagem de programação convencional como C, C++, C#, Java, Python, etc.

Para isso o VisuAlg usa uma linguagem de especificação de algoritmos baseada na língua materna dos alunos, que é o ideal para os aprendizes, como é o caso do *Portugol*[16] com grande popularidade nos meios académicos brasileiros e alguns portugueses. O algoritmo é todo escrito em Português adaptado, conhecido como Português Estruturado, o qual é interpretado por uma máquina virtual e no qual a atenção do aluno se concentra na expressão lógica da estratégia para resolver cada problema proposto.

[15] http://visualg3.com.br
[16] https://pt.wikipedia.org/wiki/Portugol

Como dito atrás, **Portugol** é uma linguagem imperativa com comandos, operadores, funções, estruturas de dados, e demais ingredientes para resolver problemas por computador todos escritos em língua natural (Português) para que mais naturalmente os alunos compreendam o significado de cada construtor. De modo a ilustrar esta ideia, as listagens abaixo mostram dois exemplos de algoritmos clássicos codificados em pseudo-código Portugol (Silva, 2006).

Exemplo de um programa que utiliza a estrutura "para".

```
algoritmo fatorial;

var
        res : inteiro;
        fat : inteiro;
        x : inteiro;

início
        escreval ("Digite um número:");
        fat <- leia();
        res <- 1;
        para x de fat até 1 passo -1 faça
                res <- res * x;
        fimpara
        escreval("fatorial de ",fat," é igual a ",res);
fim
```

Exemplo de algoritmo que utiliza funções.

```
algoritmo fatorial_recursivo;

var
        x : inteiro;
início
        escreval ("Digite um número:");
        x <- leia();
        escreval ("fatorial de ",x," é igual a ",fatorial(x));
fim



função fatorial (z:inteiro) : inteiro

início
        se z = 1 então
                retorne 1;
        senão
                retorne z * fatorial(z-1);
        fimse
fim
```

Para interpretar o Portugol, além do VisuAlg que se está a apresentar, existem vários outros interpretadores, como por exemplo: o Portugol Studio[17], ambiente este que possui uma sintaxe fácil, exemplos, material de apoio à aprendizagem e permite a criação de jogos; o G-Portugol[18] (Silva, 2006); ou mesmo o Portugol IDE[19] (Manso et al., 2015) desenvolvido no Politécnico de Leiria e que é um ambiente de aprendizagem direcionado para o ensino superior e que permite editar e animar os algoritmos escritos em Portugol oferecendo dois editores alternativos, um textual e outro que suporta uma linguagem gráfica baseada nos fluxogramas. Estas linguagens permitem realizar as operações necessárias para a codificação de algoritmos simples, são compatíveis entre si e é possível alternar entre as duas nas fases de edição, execução e depuração (Manso et al., 2015; Manso et al., 2009). Na Figura 3.16 é possível ver a linguagem algorítmica e a sua execução passo a passo. Os alunos além de seguirem o fluxo de execução das instruções, podem visualizar as variáveis e suas alterações. Na Figura 3.17, é representa o mesmo algoritmo, mas em fluxograma.

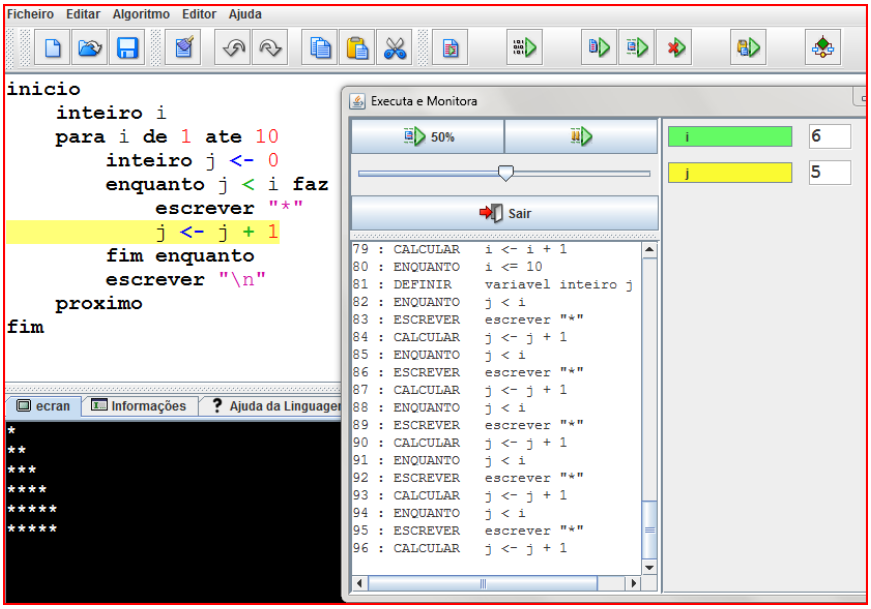


Figura 3.16: Portugol IDE desenvolvido no IPS/Tomar (interface/editor textual)

[17] http://lite.acad.univali.br/portugol/

[18] http://www.ebah.pt/content/ABAAABUdkAG/manual-g-portugol

[19] https://vinyanalista.github.io/portugol/

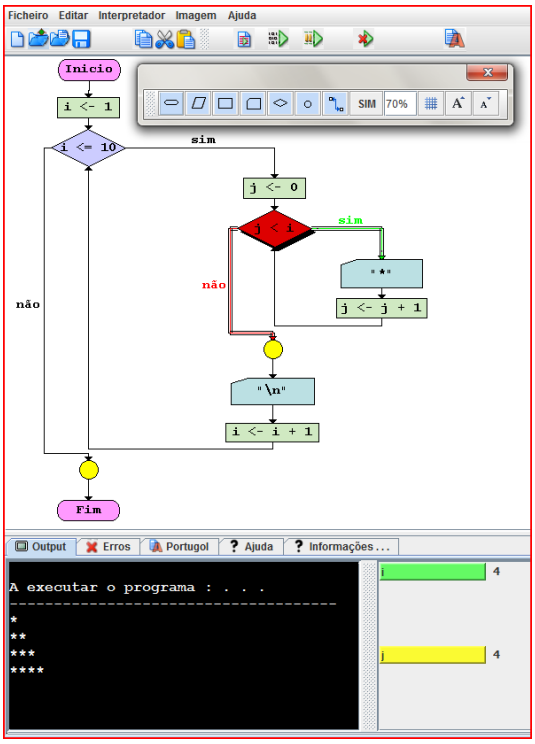


Figura 3.17: Portugol IDE desenvolvido no IPS/Tomar (interface/editor gráfico)

Como se disse atrás, VisuAlg é um ambiente de animação de algoritmos com grande popularidade actual no Brasil baseado no Portugol que permite criar, editar, executar os algoritmos em Português estruturado como se fosse um "programa" de computador.

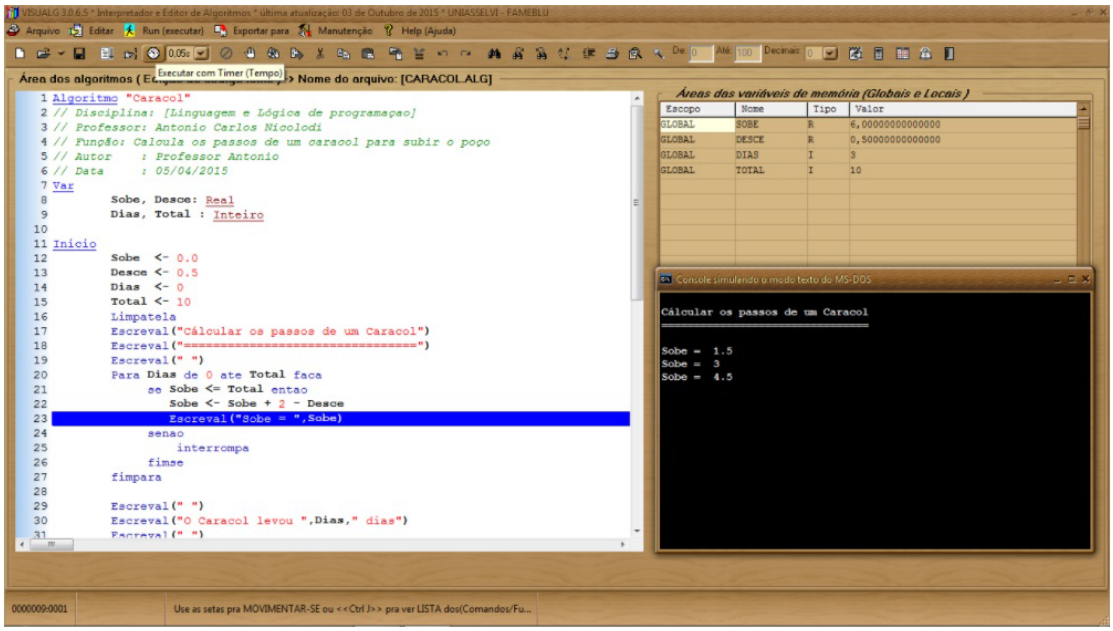


Figura 3.18: Janela do Visualg

Na Figura 3.18, é possível ver a janela do VisuAlg em que do lado esquerdo da janela se encontra o editor de texto onde se pode escrever o código fonte do algoritmo. No lado superior direito existe uma janela onde se vê o conteúdo das variáveis que estão em memória e por último ainda tem uma janela que permite ver a saída gerada durante a execução do algoritmo.

Ferramentas como estas (VisuAlg e Portugol-IDE) estão a ser cada vez mais adotadas porque permitem melhorar a eficácia do método tradicional de desenvolvimento e verificação de algoritmos.

3.1.11 Sistema Python Tutor

Pyton Tutor[20]: Online Python Tutor é uma ferramenta de visualização de programas baseadas na web para Python, em que o programa é executado dentro de um navegador da Web sem nenhum *software* necessário ou instalação de plugin (*software* livre e de código aberto, disponível em pythontutor.com). Esta ferramenta permite aos alunos ultrapassar as dificuldades na aprendizagem da programação, ou seja, entender o que acontece quando o computador executa as linhas do código. Professores e alunos podem escrever programas em diversas linguagens, Python, Java, JavaScript, Ruby, C e C++ diretamente no browser da web e visualizar passo a passo como o programa é executado. Além da vantagem de ser possível ver o fluxo de execução do programa e seus resultados internos (a nível das variáveis) e externos (a nível das saídas produzidas), e também ser possível partilhar uma sessão com uma outra pessoa online e discutir o respetivo comportamento do programa, os utilizadores podem avançar ou retroceder na execução do programa para visualização do estado das estruturas de dados e partilhar essas visualizações na web (Baldwin, 2015; Guo, 2013).

[20] http://www.pythontutor.com

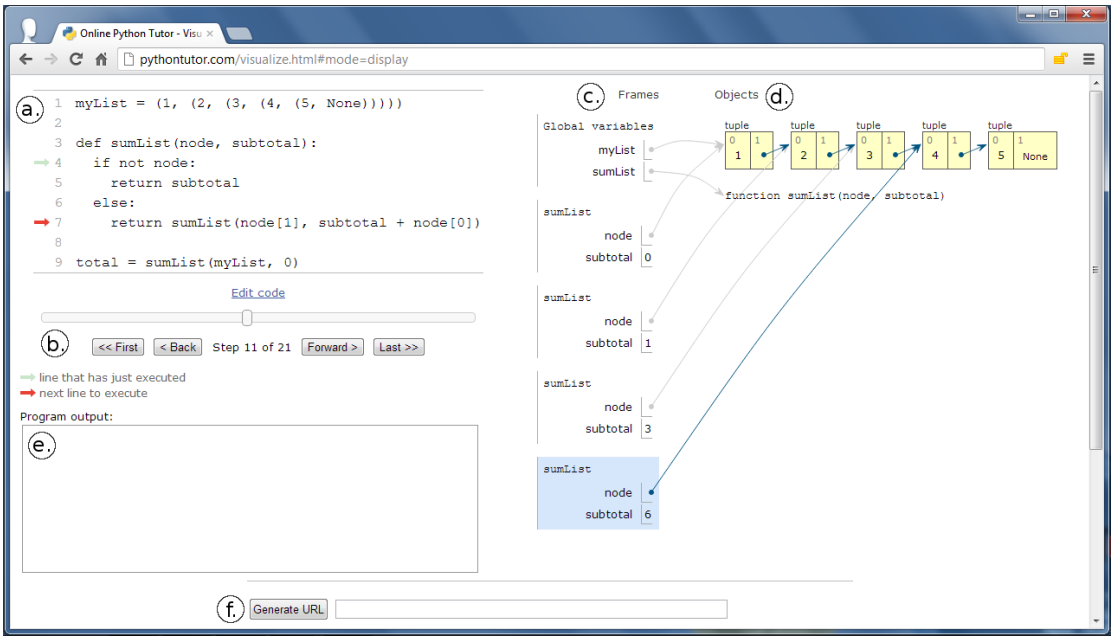


Figura 3.19: Página web de Online Pyton Tutor

A Figura 3.19 mostra uma imagem da página web de Online Python Tutor. Nela, estão representadas algumas linhas de código na linguagem Python cujo output é a soma dos elementos de uma lista podendo ser visualizado o passo 11 de 21.

Os componentes correspondentes a cada etiqueta (de 'a' a 'f') da Figura 3.19 são:

a) O quadrante mostra o código do programa, sendo que a seta vermelha aponta para a próxima linha a ser executada (exemplo da linha 7) e a seta verde aponta para a linha qua acabou de ser executada para ajudar o utilizador a seguir o fluxo de controlo.
b) A barra indica o ponto de execução em determinado momento e cada ponto representa uma única linha executada, podendo o utilizador mover a posição da barra para saltar para determinado ponto. Também é possível usar os botões para o mesmo efeito.
c) Este painel mostra vários quadros sendo possível ver quais as variáveis globais e uma representação da lista no ponto de execução, em que a lista vai crescendo para baixo. Cada quadro tem o nome da função e a lista das variáveis locais, em que o valor de cada uma das variáveis é uma seta que aponta para um objeto do conjunto. O quadro ativo está representado a azul.

d) O painel de objetos mostra representações visuais de objetos Python e as referências entre si.
e) Nesta posição é representada a caixa de texto de saída do programa que mostra todas os valores de impressão emitidas pelo programa.
f) Botão para criar um URL que identificará o programa e também o ponto de execução em que se encontra. O URL codifica o conteúdo do programa e o utilizador pode enviá-lo por email ou publicá-lo num fórum de discussão.

O Python Tutor on-line pode ser incorporado perfeitamente nas páginas da web, sem nenhum *software* necessário ou instalação de plugins. Os alunos podem ler lições e interagir com visualizações de códigos dentro da mesma página. Abrir qualquer visualização no Online Python Tutor é tão simples e rápido como visitar um URL de um navegador da Web.

Uma das características fortes deste animador é a forma como permite visualizar estruturas de dados dinâmicas complexas, sendo sabido que é um dos assuntos mais difíceis para os alunos.

3.1.12 Sistema EDPVE

EDPVE--Example-Based Dynamic Program Visualization Environment (Tekdal, 2013): Mais uma ferramenta concebida para ser utilizada por alunos que frequentam o ensino da programação. Como se tem dito, o autor desta ferramenta também afirma que a aprendizagem da programação é um processo difícil e complicado. Uma das principais razões é que os alunos têm dificuldade em entender o que o computador está realmente a fazer quando executa uma linha do programa e o que acontece ao mesmo tempo na memória.

A Visualização de Programas (VP), segundo este autor, pode ser dividida em visualização dinâmica do programa (VDP) e visualização estática do programa (VEP), referindo-se ao comportamento e à estrutura do programa, respectivamente. A VDP mostra a dinâmica de execução do programa para análise, compreensão do que se passa durante a execução do programa e debugging de erros. Por outro lado, as ferramentas VDP são úteis para produzir representações textuais a fim de aumentar a compreensão do código do programa.

Assim, foram desenvolvidas ferramentas de visualização de programas e animação para mostrar, por exemplo, a execução das instruções do programa, utilizando efeitos gráficos. A

interface do sistema EDPVE é constituída por seis janelas como visualizado na Figura 3.20. Estas janelas incluem controlo de animação, código do programa, Fluxograma, Variáveis, Console e Barra de status. Através da janela de controlo de animação, o utilizador pode iniciar, parar, reiniciar a execução do programa, executando as instruções passo-a-passo ou de forma contínua e ajustar a velocidade da execução do programa. Quando se dá inicio à execução do programa, a visualização das linhas do código do programa e da janela do fluxograma são destacadas ao mesmo tempo, tendo o utilizador a opção de fechar uma janela e observar a execução do programa na outra. Durante a execução do programa, os comentários sobre a linha atual são exibidos na barra de status e os valores das variáveis são listadas na janela Variáveis. A janela Console aparece durante a execução dos comandos de entrada e saída para uma melhor interação simulando o resultado da execução do programa.

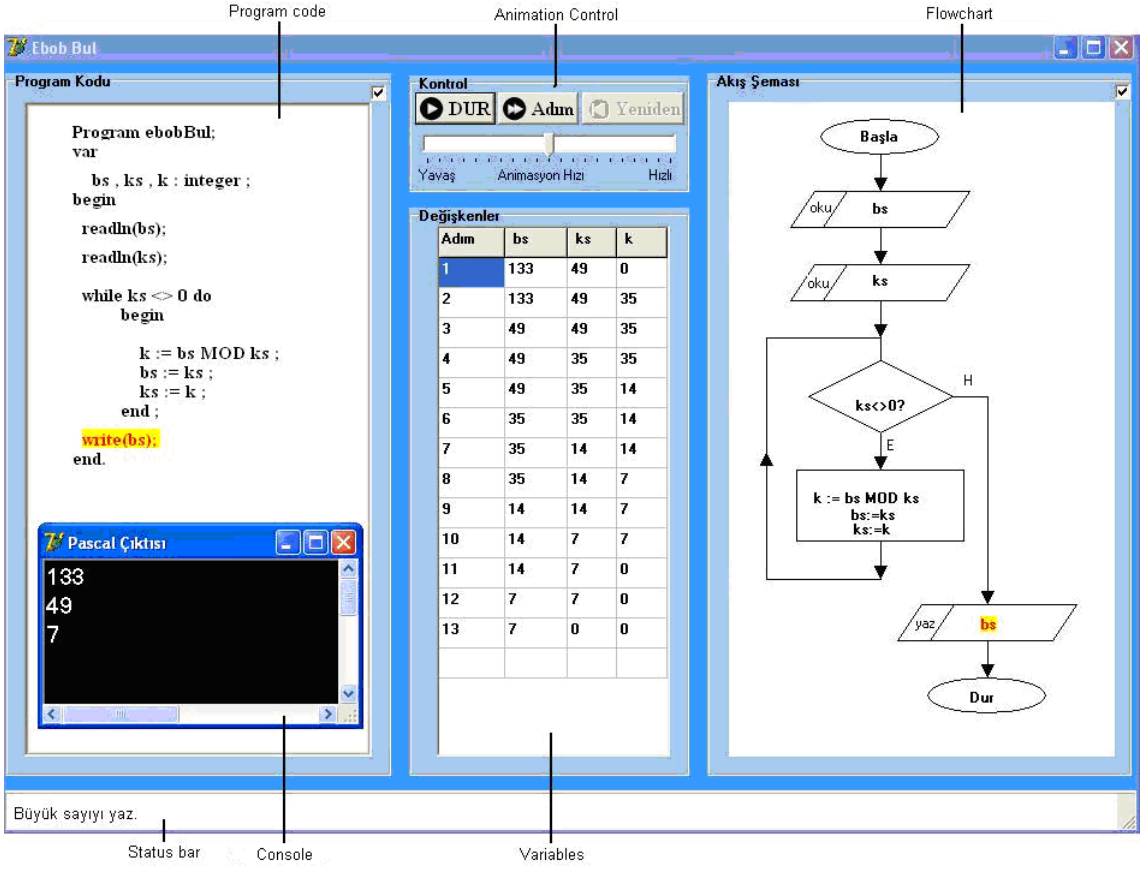


Figura 3.20: Interface do EDPVE

Mehmet Tekdal (Tekdal, 2013) realizou então um estudo com o objetivo de investigar e comparar o efeito dos dois sistemas de aprendizagem assistidos por computador, o EDPVE (Example-Based Dynamic Program Visualization Environment) e o ESPVE (Example-Based Static Program Visualization Environment) (Figura 3.21). Como era de esperar, houve uma

diferença significativa entre os experimentos da ferramenta EDPVE e ESPVE em favor da EDPVE. Este resultado no teste de desempenho sugere que os alunos que utilizaram o ambiente dinâmico (EDPVE) apresentam melhores resultados do que os alunos que usaram o ambiente estático (ESPVE). A conclusão do estudo é consistente com uma série de pesquisas anteriormente realizadas sobre a utilização de técnicas de visualização de programas, que concluíram que a animação (baseada na visualização dinâmica) melhora o desempenho na aprendizagem do aluno. O resultado deste estudo serviu para provar que o ensino e a aprendizagem de programação pode ser melhorada através do uso de técnicas dinâmicas de visualização de programas.

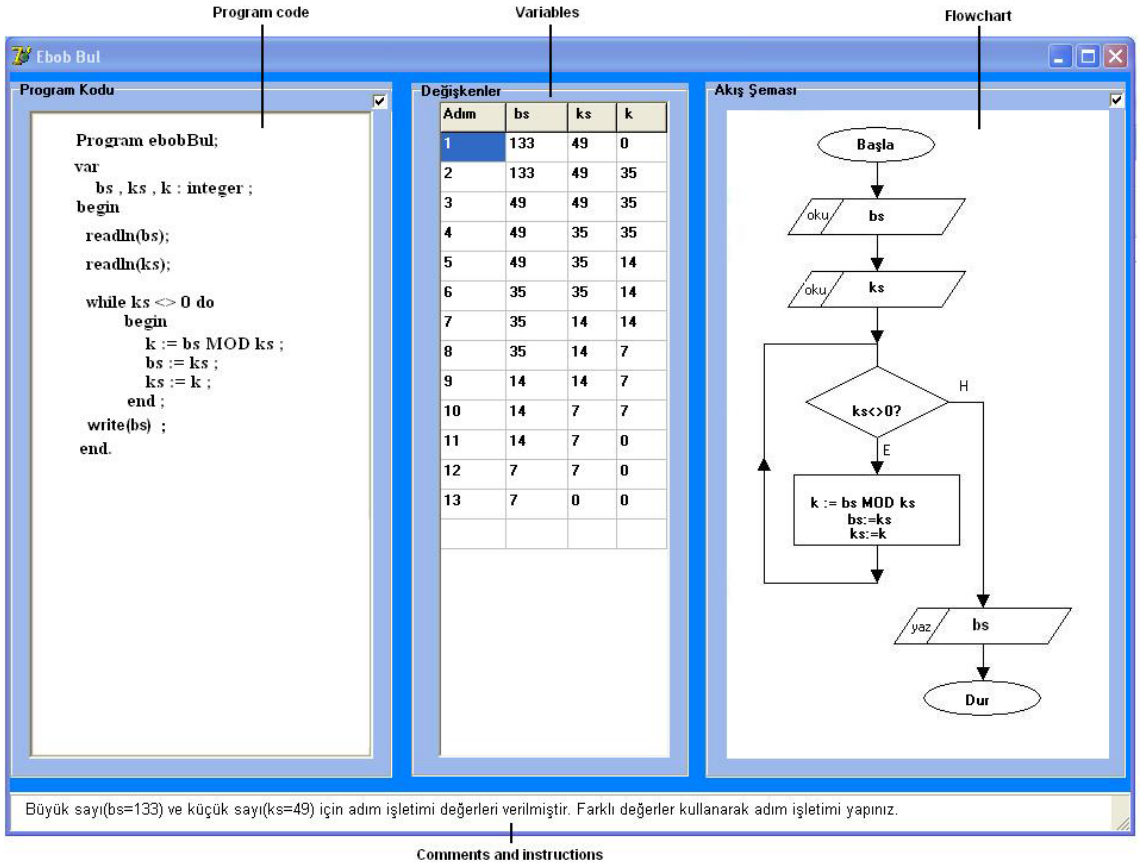


Figura 3.21: Interface do ESPVE

3.1.13 Sistema VIP

VIP[21]: é uma ferramenta de visualização para ajudar os alunos a analisar graficamente programas escritos em C ou C++, pois C é ainda uma das linguagens mais usadas para a

[21] http://www.cs.tut.fi/~vip/en/

aprendizagem da programação (Kirby et al., 2009). O professor pode explicar ao alunos o funcionamento das instruções em determinados pontos da execução.

Como o VIP se destina a alunos da introdução à programação, é importante considerar os seguintes requisitos:

1. Ser fácil para o professor criar novas visualizações;
2. Ser fácil para o aluno de usar e de aprender;
3. Ter uma interface apelativa;
4. Fornecer explicação aos alunos das diferentes etapas de execução;
5. Permitir editar o código para que os alunos possam experimentar livremente;
6. Suportar uma sintaxe da linguagem que seja um subconjunto de C++;
7. Ser acessível a partir da web.

A ferramenta foi projetada para visualizar qualquer código fonte automaticamente, sem necessidade de especificar explicitamente como as construções da linguagem devem ser visualizadas (Virtanen et al., 2005).

A Figura 3.22, mostra um ecrã da ferramenta VIP. No canto superior direito é possível ver o painel de avaliação que mostra o histórico de todas as instruções já executadas. No canto superior esquerdo, o professor pode incorporar informação que acha importante para ajuda do aluno. Nas restantes janelas inferiores é possível ver o código fonte em C++, os valores que as variáveis vão assumindo e o respetivo output produzido durante a execução (Sorva et al., 2013).

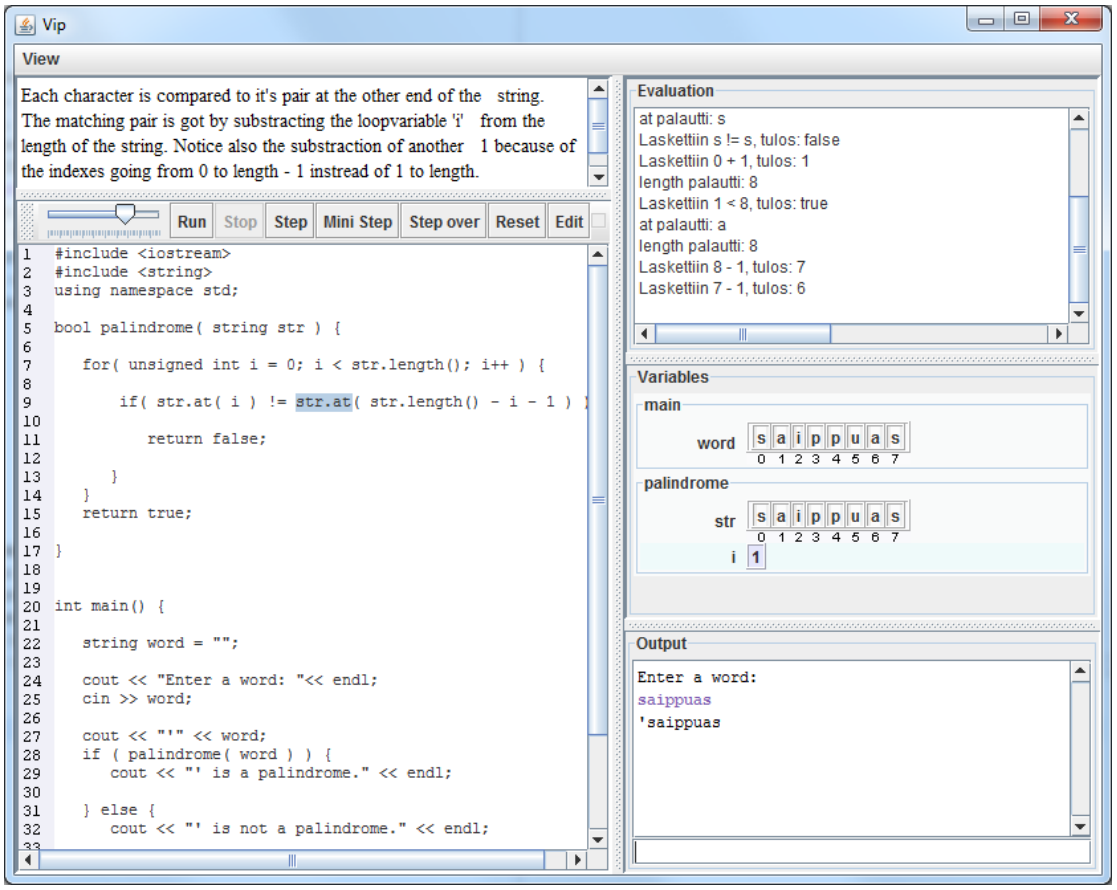


Figura 3.22: Janela do Sistema VIP

Dada a semelhança desta ferramenta com todos os animadores anteriormente descritos, não pareceu muito relevante entrar em mais detalhes.

3.2 Ambientes de Programação Visual

Todas as ferramentas faladas na secção anterior estão relacionadas com a visualização ou animação de programas desenvolvidos em linguagens de programação tradicionais. Nesta secção serão apresentados alguns ambientes de programação menos convencionais que permitem editar, executar e visualizar os programas em linguagens visuais, concebidas com o propósito de facilitar o ensino da programação.

3.2.1 Ambiente Alice

Alice2[22]: educacional para ensino de programação através de um ambiente de animação interativo 3D (OpenSource Windows e Linux 115 MB), que permite ao aluno aprender a construir programas e fazer o debug dos mesmos. O aluno pode criar o seu próprio mundo, usando modelos de objetos em 3D que povoam o mundo virtual em Alice2. Dispõe de um ambiente em que não é necessário digitar o código do programa, mas selecionar elementos com os nomes dos comandos e dos objetos, ou seja, o código é criado através da seleção de títulos que são colocados no sítio correto. Note-se que os arrastamentos e colocações podem apenas ocorrer se estiverem sintaticamente corretas.

A interface do Alice2 é ilustrada nas Figuras 3.23 e 3.24. Em Alice2, um sucessor de Alice, o código é criado através da seleção de opções e seu arrastamento para determinada posição. Como as linhas de código não podem ser editadas e os arrastamentos e colocações podem apenas ocorrer se estiverem sintaticamente corretos, não surgem erros de sintaxe e dessa forma os alunos podem construir programas sem erros de compilação. Para isso, apresenta a visualização dos objetos num contexto similar ao mundo real. Alice oculta do utilizador os detalhes envolvidos na criação de um programa real, resultando numa ferramenta conveniente para aprendizes com pouco ou nenhum conhecimento de programação (Kelleher et al., 2002).

Há cinco regiões na interface do Alice2:

1) a cena
2) a árvore de objetos
3) a área de detalhes dos objetos
4) a área da animação
5) a área de comportamentos.

[22] http://www.alice.org/

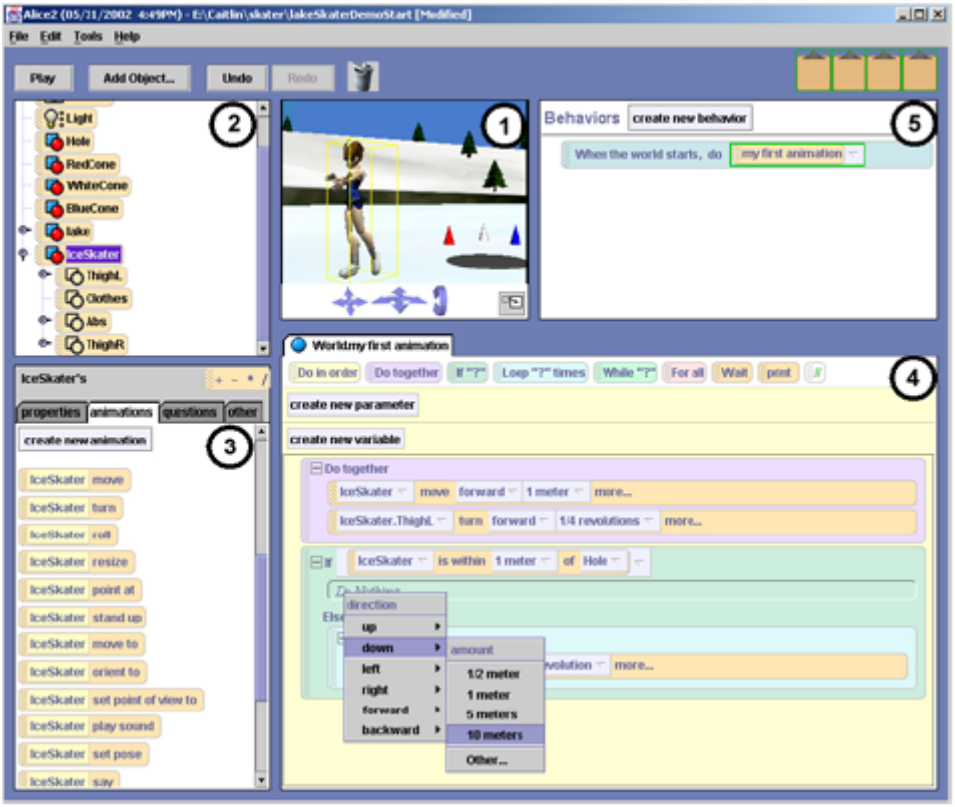


Figura 3.23: Interface do Alice2

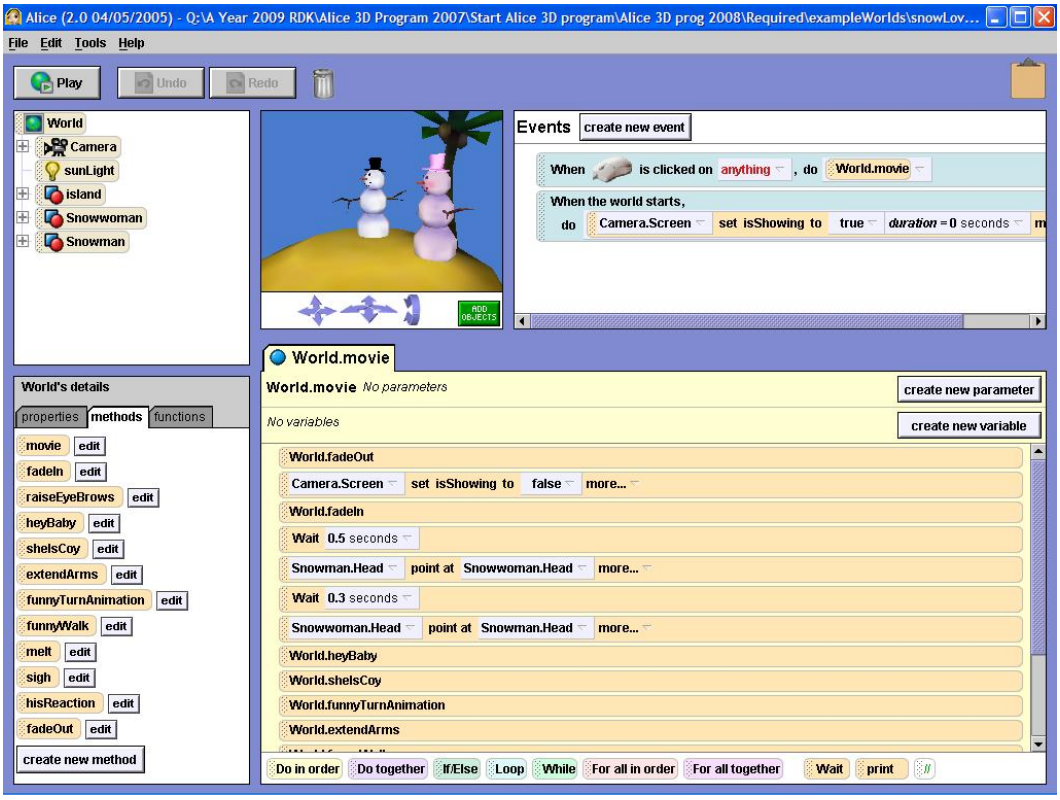


Figura 3.24: Interface do Alice

3.2.2 Ambiente Scratch

Scratch[23] : Linguagem desenvolvida no MIT para criar histórias, animações e jogos para ensinar programação e matemática a crianças a partir de 8 anos (freeware Windows e Mac OS).

Scratch é muito mais acessível que outras linguagens de programação, por utilizar uma interface gráfica que permite que programas sejam construídos à custa de blocos que se montam encaixando-os, lembrando o brinquedo LEGO. Utiliza uma sintaxe comum a muitas linguagens de programação. Cada bloco da linguagem corresponde a um comando em separado. Neste sistema o aluno pode executar um programa simples, como apresentado na Figura 3.25. Pode-se dizer que o Scratch foi inspirado na maneira como os DJs fazem as misturas de sons para criarem novas músicas, ou seja como mistura imagens, sons e outros programa.

Figura 3.25: Interface do Scratch

[23] http://scratch.mit.edu/about/

Quando abrimos o Scratch, aparecem no ecrã as seguintes áreas:

1 - Armazém de Instruções com 8 diferentes caixas de instruções

2 - Janela de Projetos onde se encontram as instruções (scripts), os fatos dos atores (costumes) e sons (sounds).

3 - Simulador onde se pode ver o resultado das instruções programadas.

4 - Área onde ficam todas os "personagens" que entram no projeto.

O Scratch é uma linguagem de programação muito simples e intuitiva, o que é recomendada para ser usada por principiantes, jovens ou adultos, que queiram iniciar-se no mundo da programação de computadores, ganhando gosto para outras linguagens mais tradicionais.

3.2.3 Ambiente Kodu

Kodu[24]: destinada principalmente para crianças, é uma linguagem de programação para a criação de jogos. O seu ambiente usa uma linguagem simples e baseada em ícones. A linguagem Kodu usa uma gramática independente de contexto, sendo uma linguagem de alto nível e visual. É inspirada pela robótica e cada uma das personagens e objetos é programado individualmente para interagir com o mundo (Ribeiro, 2012).

O ambiente de desenvolvimento em Kodu pode ser controlado usando um comando da Xbox ou o teclado e o rato do computador. Independentemente do equipamento utilizado, o menu de ajuda surge sempre no canto superior esquerdo do ecrã do programa (Figura 3.26).

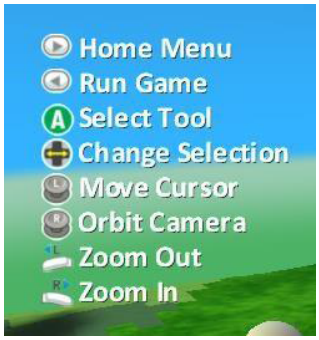


Figura 3.26: Menu de ajuda do ambiente Kodu

[24] https://www.kodugamelab.com

Kodu oferece a possibilidade de criar mundos e adicionar os elementos pretendidos a partir de um menu circular com vista a criar o jogo, como exemplificado na Figura 3.27.

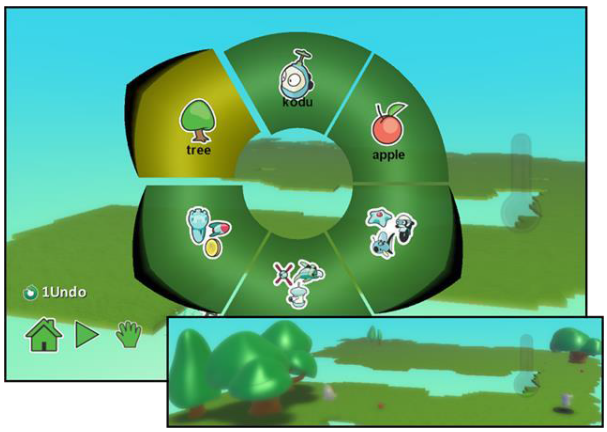


Figura 3.27: Menu circular para criação do jogo

O utilizador pode estabelecer as condições e ações das personagens obedecendo à lógica "quando-faz", que quer dizer que se deve sempre definir uma condição e uma ação para que cada personagem execute um determinado movimento ou procedimento[25].

Figura 3.28: Visualização dos eventos (condição e ação)

25 Regulamento – Kodu Kup Prtugal, Recursos para o Professor.

Segundo Stolee (Stolee, 2010) a linguagem do Kodu é orientada para eventos, em que cada linha de programação tem a forma de uma condição e uma ação (Figura 3.28). Um exemplo de uma regra bastante simples é a número 1 acima: *quando vê a maça vermelha, avança rapidamente*, em que ver a maçã vermelha é a condição, e avançar rapidamente é a ação. Um programa nesta linguagem é formado por uma série de regras de produção, onde o lado esquerdo mostra uma variável, e o lado direito contém variáveis e terminais. Segundo o autor a Linguagem Kodu pode ser representada pela seguinte gramática:

Fragmento simplificado da Gramática:

Program → Rule Program | Rule

Rule → Condition Action

Condition → Sensor FilterSet

Action → Actuator Modifier Selector

Sensor → see | ...

FilterSet → Filter FilterSet | Filter

Filter → apple | red | ...

Actuator → move | ...

Selector → toward | ...

Modifier → quickly | ...

Esta linguagem, ajuda os utilizadores no desenvolvimento e pensamento crítico e na resolução dos problemas. É de salientar que esta linguagem terá sucesso numa faixa etária baixa mas não parece a mais adequada para o ensino superior (Ribeiro, 2012).

3.3 Sumário

Quando se inicia a aprendizagem da programação é muito importante encontrar um ambiente simples e diferente dos ambientes de desenvolvimento profissionais, que ajude os alunos a perceberem o funcionamento dos programas que estão a tentar construir. Neste sentido, como verificado neste capítulo, vários autores interessaram-se pelo assunto, desenvolvendo ambientes menos complexos e mais apelativos que os ambientes profissionais, com funcionalidades importantes para aclarar e incentivar programadores pouco experientes. Estes sistemas permitem a compreensão de aspetos importantes da programação (algoritmos e programas) através da animação de pseudo-código, de fluxogramas, ou mesmo de programas escritos em linguagens genéricas, como por exemplo Pascal, C, C++, Java, Python, entre outros. Os mais interessantes e apelativos são os que permitem que os alunos introduzam e simulem os seus próprios algoritmos e programas. A animação baseada em simulação permite apresentar a visualização dinâmica do programa (isto é, durante a sua pseudo-execução) e auxilia a compreensão do aluno ao seu próprio ritmo.

Apesar das inúmeras vantagens identificadas, o seu impacto real ficou aquém do esperado, conforme se irá analisar no capítulo seguinte com o intuito de encontrar soluções que o complementem.

4. ANIMAÇÃO DE PROGRAMAS: IMPACTO

Após o estudo anteriormente apresentado, é importante analisar o impacto destas ferramentas de animação de programas segundo a opinião de três autores: Hansen, Stasko e Hundhausen.

4.1 Perspetiva de Hansen

Segundo Steven Hansen (Hansen et al., 1999), a ideia de usar animações para ilustrar o comportamento dinâmico de algoritmos é uma preocupação antiga. Vários sistemas foram construídos desde então, com a crença de que as animações serviriam como ajudas de aprendizagem efetiva dos estudantes. No entanto, só recentemente tem surgido a dúvida se de facto as animações do algoritmo realmente ajudam a aprendizagem. Infelizmente, os resultados de experimentos conduzidos por esta questão têm sido decepcionantes. Acredita-se então, que tentativas anteriores utilizando animação para ensinar o comportamento do algoritmo foram insatisfatórias não por causa de uma falha com a animação como uma técnica, mas devido à abordagem usada para transmitir as animações.

Infelizmente sobre todas as evidências, os resultados de vários estudos são, na melhor das hipóteses, uma mistura. Enquanto animações são recebidas com entusiasmo pelos alunos, nenhum dos estudos tem provado conclusivamente que estas apresentações visuais realmente melhoram a aprendizagem.

Será porque a animação é uma ferramenta de ensino ineficaz? A intuição de vários autores diz-nos que isso não é provável (Hansen et al., 1999)

Assim sendo, será necessária uma reformulação no processo de utilização pedagógica dos sistemas de animação de algoritmos/programas para melhorar a aprendizagem.

Finalmente, Hansen fornece resultados de oito estudos empíricos realizados durante três anos, envolvendo mais de 230 alunos do ensino superior para se obter uma visão geral sobre a utilização de técnicas de visualização/animação de algoritmos na Universidade de Auburn. Como se disse no inicio da secção, os resultados desses vários experimentos foram dececionantes, o que levou este autor a concluir que uma reformulação no processo é necessária para se aproveitar o poder da visualização/animação para melhorar a aprendizagem. Para este estudo foi construído um sistema chamado HalVis (Hypermedia Algorithm Visualizations). A ideia-chave é incorporar animações de algoritmo em vários níveis de abstração dentro de um ambiente de hipermídia. Estes experimentos fornecem validade estatística para a hipótese de que visualizações de algoritmo de Hipermídia podem ser significativamente mais eficazes do que os métodos de ensino tradicional e animações de algoritmo, consideradas isoladamente.

4.2 Perspetiva de Stasko

T. Stasko e colegas (Stasko et al., 1996) concordam que a animação de programas pode facilitar a aprendizagem. O uso de imagens e visualizações como auxiliares de ensino é uma prática aceite por vários autores desde há muitos anos. Os livros didáticos contêm várias imagens assim como os próprios professores usam o quadro tradicional, diapositivos ou transparências, apresentando diagramas para ajudar numa determinada explicação. A animação vai um passo além da visualização, pois parece mais capaz de explicar um processo de evolução dinâmica. Estes autores fazem uma analogia com o cinema no qual se usa animação para contar uma história. O foco destes autores é o uso da animação para ensinar

programação e ensinar aos alunos como os é que os algoritmos funcionam. Chamam animação de algoritmos ao uso de imagens para ilustrar algoritmos, programas e processos. A animação de algoritmos realmente serve dois propósitos fundamentais como auxílio educativo: fornece uma representação concreta das abstrações e operações inerentes a um algoritmo ou programa; e retrata a dinâmica de um processo que leva tempo a evoluir. Por estas razões, professores e investigadores põe a hipótese de que estes podem tornar-se valiosos auxiliares de ensino. Infelizmente, esta hipótese é amplamente baseada na intuição pois, como já Hansen (Hansen et al., 1999) o mostrou, a validação experimental não confirmou tal sucesso.

Em geral, a informação que pode ser transmitida por uma animação de um algoritmo pode ser dividida em duas grandes áreas: processual e conceptual. As animações podem ser eficazes na transmissão de ambos os tipos de informação. O conhecimento processual representa a sequência de passos realizados pelo algoritmo. Estes passos podem frequentemente ser exibidos explicitamente numa animação, e ajudar o aluno a aprender e a lembrar-se deles. O conhecimento conceptual refere-se a uma compreensão do "sentido" e das implicações do algoritmo, ou seja, pode permitir a um aluno estimar a eficiência do desempenho de um algoritmo em determinadas situações, sem ter que mentalmente executar o algoritmo. Animações podem incentivar e fazer com que seja relativamente fácil ao aluno fazer e testar previsões do que vai acontecer a cada passo do algoritmo.

Para os autores referidos avaliarem a afirmação geral de que animações podem ajudar os alunos a aprender algoritmos de forma mais eficaz, foram feitos dois estudos. Foram utilizadas Animações e instruções durante este estudo para que os alunos previssem o comportamento de um algoritmo. Os resultados das duas experiências sugerem que os benefícios de animações não são óbvios, o que levou Stasko a propor que os professores devem fazer uma análise muito cuidadosa para determinar que tópicos ou subáreas de conhecimento (itens do programa das suas disciplinas) devem ser apresentados com recurso à animação para poderem ser uma ajuda efetiva para o aluno (Stasko et al., 1996).

4.3 Perspetiva de Hundhausen

Desde o seu aparecimento no final de 1970 que a tecnologia da visualização de algoritmos evoluiu para ambientes de programação interativos que permitam aos estudantes construir rapidamente as suas próprias visualizações (Hundhausen et al., 2002). Ao longo das últimas duas décadas, essa evolução de *software* tem vindo a reforçar a ideia de uma melhoria na educação da computação.

O *software* de visualização de algoritmos foi usado, por exemplo:

- para ajudar os professores a ilustrar as operações de um algoritmo;
- para ajudar os alunos que estudam e aprendem algoritmos fundamentais;
- para ajudar os professores a encontrar bugs em programas dos alunos durante as aulas;
- para ajudar os alunos a aprender as operações básicas de um tipo abstrato de dados num laboratório de informática.

Assim como os outros autores, D. Hundhausen (Hundhausen et al., 2002), afirma que apesar de seu apelo intuitivo como um auxílio pedagógico válido, a tecnologia de visualização de algoritmos não foi capaz de ajudar no ensino da área da ciência da computação. Um número reduzido de educadores (que também desenvolvem tecnologia de visualização de algoritmos) tende a usar a sua própria tecnologia de visualização e a maioria tende a usar tecnologias pedagógicas mais tradicionais, tais como quadros e retroprojetores.

Por que é que os professores de ciência da computação tendem a não usar a tecnologia de visualização de algoritmos?

Os professores geralmente citam várias razões:

- Sentem que não têm tempo para aprender sobre o assunto em questão;
- sentem que usá-la, tiraria tempo necessário para outras atividades da aula;
- sentem que a criação de efeitos visuais na sala de aula exige muito tempo e esforço;
- sentem que não é educacionalmente eficaz.

Todas estas razões, contribuem, de alguma forma, para o insucesso da tecnologia de visualização. No entanto, a quarta razão, que defende que esta tecnologia não ajuda os alunos

a aprender algoritmos melhor do que os métodos convencionais de ensino, destaca-se como sendo particularmente importante. Na verdade, o objetivo subjacente desta tecnologia é ser educacionalmente eficaz, de modo que, certamente, não é motivo para adotar a tecnologia se não for eficaz.

A fim de entender melhor a eficácia da visualização de algoritmos, D. Hundhausen (Hundhausen et al., 2002) e seus colegas apresentaram um conjunto de 24 estudos experimentais. Estes autores procuraram fazer duas análises separadas: uma análise de variáveis independentes em que cada estudo foi vinculado a uma teoria de aprendizagem na tentativa de determinar qual ou quais tiveram mais sucesso preditivo; e uma análise das variáveis dependentes, que permitiu determinar quais as técnicas que foram mais sensíveis aos benefícios da aprendizagem da tecnologia de visualização de algoritmos.

Na primeira análise os resultados foram examinados quanto às variáveis independentes numa tentativa de compreender melhor quais os fatores que causaram mais eficácia e consistência. Por outro lado, uma segunda análise olhou para os resultados de variáveis dependentes, a fim de entender a forma como a eficácia tem sido definida e que medidas são mais capazes de deteta-la.

A tecnologia de Visualização de algoritmos é educacionalmente eficaz? Segundo Hundhausen a resposta será um “sim”.

De que forma esta tecnologia é educacional eficaz, e de que forma não é educacionalmente eficaz?

Começando com a forma não eficaz, foram encontrados alunos que não demonstraram uma melhoria significativa da aprendizagem comparativamente com alunos que usaram os materiais convencionais. Esta observação sugere que a mera presença de tecnologia de visualização não garanta que os alunos irão aprender um algoritmo. Em particular, este estudo sugere que os meios educacionais mais bem sucedidos da tecnologia de visualização de algoritmos são aqueles em que a tecnologia é usada como um veículo para envolver ativamente os alunos no processo de aprendizagem de algoritmos. Como exemplo, esta tecnologia tem sido usada com sucesso para envolver ativamente os alunos em atividades como análise de comportamentos algorítmicos, exercícios de previsão e exercícios de

programação. Em tais casos, ao invés de ser um instrumento para a transferência de conhecimento, esta tecnologia serve como catalisador para a aprendizagem. As visualizações de algoritmos são educacionalmente eficazes na medida em que habilitam os alunos a construir os seus próprios entendimentos de algoritmos através de um processo de aprendizado ativo.

Em suma, este estudo sugere que a tecnologia de visualização de algoritmos é educacionalmente eficaz, mas não da forma convencional sugerido pelo provérbio: "uma imagem vale por mil palavras". Em vez disso, de acordo com os resultados, a forma de aprender o exercício no qual a tecnologia é usada é realmente mais importante que a qualidade das visualizações produzidas. Isto não pretende afirmar que a qualidade de visualização não importa — para uma aprendizagem bem sucedida uma visualização bem projetada contribui certamente — mas sim que a forma de atividade é mais importante do que a forma de visualização (Hundhausen et al, 2000).

Neste contexto, Peter Brusilovsy (Brusilovsky et al., 2004) mostra três caminhos complementares que tornam a visualização uma mais valia para a educação:

- visualização orientada para o envolvimento;
- visualização explanatória;
- visualização adaptativa.

O autor afirma que o maior problema da visualização é a sua natureza passiva, ou seja, o papel do aluno é apenas ver o que está a acontecer no ecrã. Mas segundo várias teorias, o aluno tem que ser ativamente envolvido no processo educativo para que a aprendizagem aconteça.

No primeiro ponto, o aluno deve ser envolvido na visualização e trabalhar mais ativamente, construindo a visualização inteira em vez de ser apenas um espectador. Convidar o aluno a prever os resultados da animação pode se tornar muito eficaz.

A visualização explicativa (*explanatory visualization*) tem como principal argumento a dificuldade dos alunos em aprender com a visualização, pois não entendem o que veem, o que aconteceu e porque aconteceu. A ideia da visualização explicativa é a cada passo da visualização haver uma explicação, por parte do aluno, numa linguagem natural. As explicações ajudam o aluno a entender o que vê.

A Visualização adaptativa (*adaptive visualization*) é baseada na suposição de que um aluno pode ter níveis diferentes de conhecimento, de diferentes elementos, de um algoritmo que está a visualizar. O aluno pode compreender alguns passos de um algoritmo melhor do que outros, logo, a visualização regular que anima tudo com o mesmo nível de detalhes pode ser uma abordagem errada. Detalhes desnecessários para uma determinada visualização podem distrair o aluno.

A ideia da visualização adaptativa é fazer coincidir o nível de detalhe da visualização de cada construção com o nível de conhecimento do aluno, ou seja, quanto menor o nível de compreensão de um aluno maior o nível de detalhe na visualização. À medida que o aluno aumenta o seu conhecimento, o nível de visualização deve diminuir, permitindo assim ao aluno se concentrar em aspetos menos compreendidos e ainda ser capaz de entender toda a visualização.

É interessante notar que muitos dos sistemas de visualização/animação mais recentes e bem sucedidos que foram descritos no Capítulo 3, seguem uma ou várias destas 3 sugestões.

4.4 Sumário

O uso de sistemas de animação para fins educacionais, é de extrema importância para a formação dos alunos, ajudando-os a entenderem disciplinas no início do curso que são pré-requisitos necessários para disciplinas mais avançadas. A animação torna-se um facilitador do processo de aprendizagem, visto que a apresentação de conceitos abstratos torna-se mais viável e didática, melhorando a qualidade do material da aula conforme se tinha procurado mostrar com o alargado número de sistemas que nas últimas décadas tem sido desenvolvidos para suportar esta abordagem.

Neste capitulo mostrou-se que o sucesso desta prática pedagógica, quando comprovado através de experimentos especialmente desenhados para o efeito, não foi tão grande como se esperava intuitivamente. Os três estudos aqui revistos, alertam os professores para os cuidados que se devem ter ao adoptar as referidas ferramentas, evidenciando-se três grandes linhas: envolver o mais possível os alunos no processo, evitando que tomem um papel de meros espectadores; completar cada passo do processo de animação com explicações

cuidadas do que foi visualizado; permitir que os sistemas se adaptem aos alunos, variando o nível de detalhe consoante o seu ritmo de aprendizagem.

Quando existe uma boa base por parte do aluno, o recurso a ferramentas didáticas como as aqui apresentadas faz com que o rendimento e desempenho aumentem proporcionando melhores resultados, melhores currículos e melhores profissionais (Santos et al., 2006).

5. AVALIAÇÃO AUTOMÁTICA DE PROGRAMAS: SISTEMAS

É muito importante dar aos alunos a oportunidade de praticar, ou seja, poder resolver exercícios de programação por si mesmos sendo nesse caso o *feedback* imediato essencial para a aquisição de conhecimento. Diferentes estudos mostram que é fundamental a existência de sistemas que permitam aos alunos desenvolver autonomamente habilidades de programação para melhorar significativamente o seu desempenho (Verdú et al., 2011).

A verdade é que nos tempos atuais os concursos de programação com avaliação automática e *feedback* estão, cada vez mais, a tornarem-se atividades importantes para a prática da programação (Verdú et al., 2011). Actualmente, já existem na Web várias plataformas de avaliação onde os alunos podem fazer o download de problemas de programação, apresentar as suas soluções e depois serem avaliados pelo sistema. É o caso de Mooshak (Leal et al., 2008), um sistema de gestão de concursos de programação baseada na Web.

Com base nessas ferramentas de apoio aos concursos, novas ferramentas surgiram para serem utilizadas em atividades de ensino, permitindo aos alunos incorporar testes no seu trabalho. Estas ferramentas aumentam o nível de satisfação e motivação dos alunos. De acordo com professores e alunos, o *feedback* deve ser fornecido e detalhado o mais

rapidamente possível. Estas ferramentas não substituem o professor, mas fornecem ajuda e aumentam o tempo em sala de aula. Os professores devem ser capazes de selecionar os problemas que pretendem apresentar aos alunos de acordo com o seu nível de dificuldade (Verdú et al., 2011).

Com as ferramentas de *software* adequadas, a correção do programa pode ser completada com a indicação da qualidade de acordo com um conjunto de métricas. Não é fácil encontrar uma abordagem única para um problema de avaliação de um trabalho de programação. Os diferentes professores podem adotar estratégias diferentes, dependendo de suas metas e objetivos, especialmente do seu próprio estilo e preferências (Joy et al., 2005). Então, o problema está relacionado com os recursos necessários para gerir a avaliação de exercícios práticos. Com esta nova abordagem e ao contrário da clássica, os alunos recebem *feedback* preciso no momento certo para benefício da sua aprendizagem.

A maioria das ferramentas disponíveis para esta finalidade incluem um subsistema de apresentação para fazer upload dos trabalhos do aluno e outra para a sua avaliação automática. O objetivo final é fornecer novas estratégias de aprendizagem para motivar os alunos e tornar a programação mais acessível e um desafio atraente.

Segundo E. Verdú e colegas (Verdú et al., 2011), é possível classificar as ferramentas de aprendizagem inicial de programação em vários tipos entre as quais, ferramentas de visualização e sistemas de submissão com avaliação automática. Como é difícil encontrar uma ferramenta que incorpore as diferentes vantagens de cada uma, para a aprendizagem da programação é importante poder fornecer, pelo menos, uma combinação de um ambiente virtual de animação e um sistema de submissão. O primeiro fornece suporte para a resolução de problemas sendo muito importante para um estudo individual e de auto-aprendizagem e o segundo suporta variados formatos de submissão de perguntas e procede à sua avaliação. No seu conjunto, fornecem ao aluno um sistema de avaliação automática e de *feedback*.

Receber *feedback* é muito importante para a aquisição de conhecimento, independentemente da estratégia de aprendizagem particular, e motiva os alunos. Contudo, dar *feedback* individual a todos os alunos de uma sala de aula é uma tarefa praticamente impossível para os professores por ser um processo bastante demorado e as turmas terem cada vez mais alunos, levando a sucessivos fracassos da estratégia (Queirós et al., 2015). Para colmatar este

problema é que existe um número importante de sistemas de submissão *on-line* que suportam a avaliação automática para os problemas de programação, conforme se disse acima. Como resultado os alunos podem praticar e melhorar suas habilidades de programação de forma independente (Queirós et al., 2012).

Outo exemplo destes sistemas, desenvolvidos para que os alunos fossem capazes de exercitar os seus programas, surge no contexto do projeto EduJudge (Verdú et al., 2011) que combina uma ferramenta de avaliação de programas *on-line* com uma base de dados significativamente grande de problemas de modo a providenciar funcionalidades pedagógicas para o ensino da programação.

Estudos feitos permitem afirmar que um dos resultados mais interessantes no uso destas ferramentas é que o nível de satisfação e motivação dos alunos não depende apenas do seu conhecimento ou de competências a nível informático. Segundo professores e alunos, o *feedback* automático deve ser melhorado, deve ser rápido e o mais detalhado possível. Esse *feedback* deve oferecer uma resposta mais pormenorizada para melhorar a qualidade de um programa, se este for mal codificado ou muito complexo. Estas ferramentas não substituem o papel do professor, mas ajudam e valorizam o tempo na sala de aula[26]. Os problemas propostos têm diferentes níveis de dificuldade, sendo útil para um aumento da compreensão da programação por parte dos alunos. Os professores devem poder então selecionar os problemas que pretendem apresentar aos estudantes de acordo com seu nível de dificuldade (Verdú et al., 2011).

Como referido anteriormente, no ensino da programação de computadores pode ser interessante e importante a avaliação automática. Com as ferramentas de *software* apropriadas a correção do programa pode ser determinada, juntamente com uma indicação da qualidade de acordo com um conjunto de métricas. Além disso, permite a detecção de plágio sendo já uma parte integrante de algumas destas ferramentas que suportam esta avaliação. Não pode existir uma abordagem única e correta para o problema da avaliação dos trabalhos de programação. Diferentes professores podem adotar diferentes estratégias, dependendo dos seus objetivos específicos e objetivos do curso, e principalmente do seu próprio estilo e preferências (Joy et al., 2005). Daí que não se pretenda que a avaliação geral e

[26] http://www.saocarlos.usp.br

final dos alunos seja feita automaticamente por ferramentas computacionais; essa tarefa será sempre deixada a cargo do professor. Então, o problema diz respeito aos recursos necessários para gerir a correção dos exercícios práticos, de modo a que os alunos recebam *feedback* preciso e no momento certo para que a sua aprendizagem possa ser beneficiada. A maioria das ferramentas disponíveis para o efeito inclui a entrega de trabalhos e avaliação automática. Isto é apropriado para uma aprendizagem inicial onde o conhecimento e a compreensão estão a ser testados. O objetivo final é fornecer novas estratégias de aprendizagem para motivar os alunos e poder afirmar que a programação pode ser um desafio fácil e atraente.

O processo de avaliação de um programa inclui três componentes principais (Figura 5.1): a primeira, correção (*correctness*), tem a ver com determinar em que medida o funcionamento do programa corresponde ao da sua especificação; a segunda, estilo (*style*), consiste e analisar o tipo de escrita seguido no desenvolvimento do programa e verifica o uso de boas práticas na escrita de um programa; a terceira componente, autenticidade (*authenticity*), diz respeito às tarefas administrativas, incluindo a verificação da identidade do aluno e verificação de plágio.

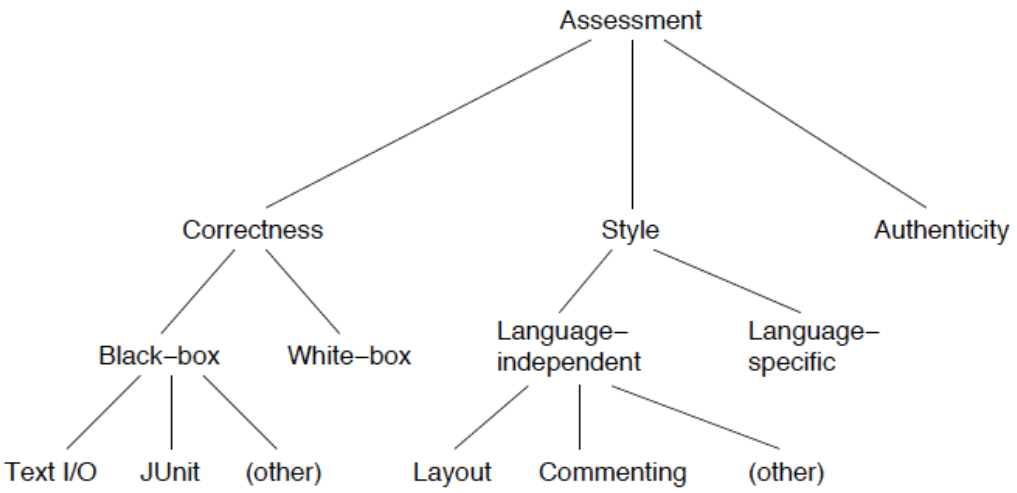


Figura 5.1: Processo de avaliação (Joy et al., 2005)

Sendo a correção do código um critério de avaliação, é frequentemente encarada pelos alunos com uma importância reduzida, pois para os alunos muitas das vezes o importante é se o programa corre ou não ("O meu programa funciona?"), em contraste com requisitos mais importantes, como por exemplo os detalhes algorítmicos que garantem que o programa

produz sempre os resultados esperados e não bloqueia em situações fora do domínio. Uma possível forma de testar a Correção de um programa pode ser conseguida especificando a saída esperada de um programa para cada entrada de um dado conjunto de testes. Descrito este mapeamento entre entrada-saída específico de cada caso, o programa estará correto se produzir a resposta certa para cada valor de entrada.

Segundo Mike Joy, Nathan Griffiths and Russell Boyatt (Joy et al., 2005), parece não haver acordo num conjunto de critérios que podem caracterizar um programa de "bom", pois diferentes académicos e profissionais de *software* sublinham diferentes aspectos no processo. No entanto, os autores afirmam ser possível identificar critérios que são comuns:

- Código comentado: uma boa prática pois os programas devem ser bem documentados no próprio código, a fim de garantir que o programa seja compreensível e de fácil manutenção por terceiros.
- Estilo do código: prática que determina que os programas devem ter um layout claro, com escolhas significativas para os nomes das variáveis e nomes de métodos.
- Código Válido: os programas devem estar corretos, tanto na sintaxe como na semântica da linguagem utilizada.
- Estrutura do código: há normalmente muitas formas alternativas de escrever um programa para uma determinada tarefa. Os alunos devem estruturar o código em módulos ou classes apropriadas.
- Uso de bibliotecas externas: muitas linguagens de programação têm bibliotecas externas de funções para a realização de tarefas específicas. Os alunos devem, sempre que permitido, fazer uso efetivo dessas bibliotecas.
- Documentação: os programas devem ter um manual de utilização assim como um documento técnico (relatório de desenvolvimento).
- Escolha e eficiência do algoritmo: existem muitos métodos alternativos para programar uma solução de uma determinada tarefa. Os programas devem usar algoritmos adequados e eficientes.
- Eficiência de código: A implementação do algoritmo escolhido deve ser eficiente, e devem ser usadas construções apropriadas da linguagem quer ao nível das estruturas de controlo, quer ao nível das estruturas de dados.

Nem todos estes critérios são aplicáveis a todos os programas e cada professor gosta de lhe dar pesos e relevância diferentes em função do contexto. Por isso, é importante desenvolver ferramentas para apoiar a aferição dos critérios que podem ser medidos automaticamente (como para avaliar a correção do programa), deixando os outros à responsabilidade do professor (Joy et al., 2005).

Após esta introdução ao processo de avaliação dos alunos de programação com base nos programas que submetem para resolver os exercícios propostos nas aulas e ao valor que o retorno da informação associada a esse processo de correção aporta para a aprendizagem, no resto deste capítulo serão apresentados alguns dos sistemas atualmente disponíveis para verificar automaticamente a correção dos programas e fornecer *feedback* ao utilizador, sistemas esses que serão designados AAP (Avaliação Automática de Programas).

5.1 Sistemas de AAP

Como exemplos de Ferramentas para Avaliação Automática de Programas podem citar-se o Boss, o Mooshak, o EduJudge e o Codeboard, cujo principal objetivo, além de testar as respostas dos alunos face a um conjunto de dados de entrada e dar uma avaliação, é permitir ao professor motivar os alunos através de um *feedback* preciso e rápido.

5.1.1 Sistema BOSS

BOSS Online Submission System[27] (Heng et al., 2005), foi desenvolvido ao longo de vários anos como uma ferramenta para facilitar a submissão on-line e a avaliação de trabalhos de programação. Ferramenta de gestão do curso desenvolvido pelo Departamento de Ciências da Computação da Universidade de Warwick, BOSS é um sistema de submissão que auxilia na avaliação dos alunos de um curso através de recolha de submissões, realização de testes automáticos para correção e qualidade, verificando se há plágio, através de uma interface que no fim retorna *feedback*. Suporta uma variedade de estilos e estratégias de avaliação e fornece o máximo de apoio aos professores e alunos. Neste contexto, o professor tem acesso a ferramentas automáticas, versáteis e flexíveis, para o auxiliar no processo de avaliação.

[27] http://www.dcs.warwick.ac.uk/boss

A versão original do BOSS foi criada no início de 1990 com uma interface gráfica e respetivas ligações a bases de dados centrais. Devido à natureza do seu desenvolvimento, este sistema tornou-se difícil de gerir e expandir, tendo sido necessário uma versão atualizada do sistema. O desenvolvimento do novo BOSS começou em 1999 e foi implementado em outubro de 2000. O BOSS atualizado foi projetado para linguagens de programação modernas e para práticas de avaliação atuais. Esta nova versão de 2001, foi codificada em Java, permitindo uma compatibilidade maior entre plataformas. Este modelo separa as funcionalidades por três servidores, tendo interface para dois tipos de utilizadores, uma para estudantes e um outro para o docente (ver Figura 5.2). Inicialmente o desenvolvimento era apenas focado na funcionalidade da submissão do aluno. Avaliação e teste seguiu-se mais tarde. Os três servidores são:

- Servidor do estudante: aceita inscrições dos alunos do curso e suporta a sua gestão;
- Servidor do docente: acessível apenas pelo docente para fins de avaliação;
- Servidor de teste: usado pelo aluno e staff para correr automaticamente os testes.

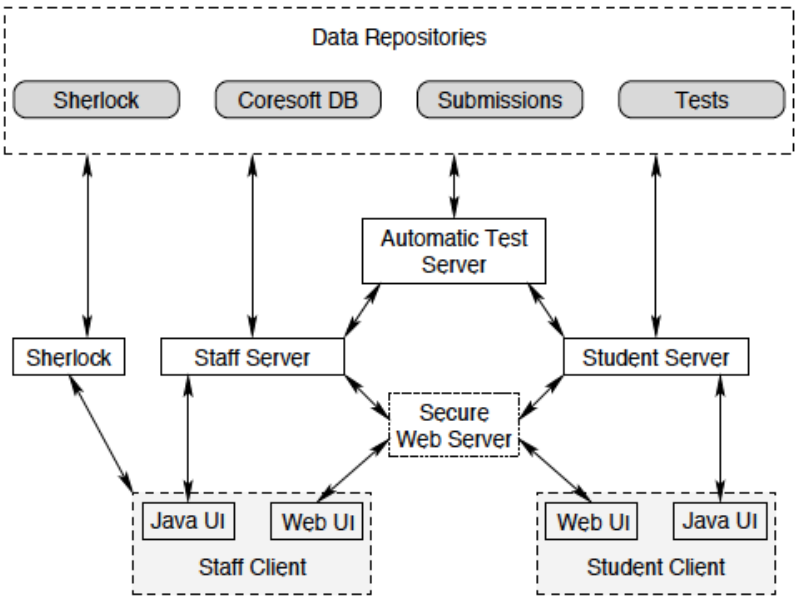


Figura 5.2: Visão geral da arquitetura BOSS (Heng et al., 2005)

Dividindo as funcionalidades entre os servidores, os benefícios técnicos e organizacionais foram grandes. Assim é garantido que os problemas com os sistemas de avaliação não afetam diretamente as submissões. Além disso, o servidor do aluno e cliente têm acessos restritos e

funcionalidades limitadas para reduzir o risco de uma falha de segurança. Um componente de detecção de plágio chamado Sherlock foi adicionada ao Boss, para auxiliar na análise dos trabalhos. A detecção de plágio pode funcionar em dois modos, um para o código fonte do programa e outro para os trabalhos redigidos em linguagem natural (Heng et al., 2005).

Um problema específico diz respeito aos recursos necessários para gerir a avaliação dos exercícios práticos, de modo a que os alunos recebam *feedback* preciso e oportuno beneficiando assim no seu progresso e desempenho.

Como já referido atrás, BOSS permite aos alunos apresentar trabalhos on-line com segurança, contendo um conjunto de ferramentas que permitem que o responsável identifique tarefas on-line. Desde o início, foi de extrema importância garantir que BOSS fosse educacionalmente sólido. Em particular, foi necessário assegurar as seguintes propriedades :

- Precisão: os dados fornecidos pelo BOSS necessitam de ser precisos.
- Adequação à finalidade: BOSS foi concebido como uma ferramenta para submissão on-line e gestão de um curso ou de disciplinas. Não pretendia ser um sistema CAL (Aprendizagem Assistida por Computador), e não deve, portanto, afetar a experiência de aprendizagem dos alunos.
- Adaptabilidade: para os utilizadores, fornece módulos de diferentes maneiras, tanto pedagogicamente como estruturalmente. BOSS não deve afetar as decisões profissionais de um professor sobre a entrega de seu material, e ainda deve ser capaz de ser usado e de ser útil para eles.

As Figuras 5.3 e 5.4 representam as janelas que dizem respeito ao estudante e ao docente respetivamente após o seu login.

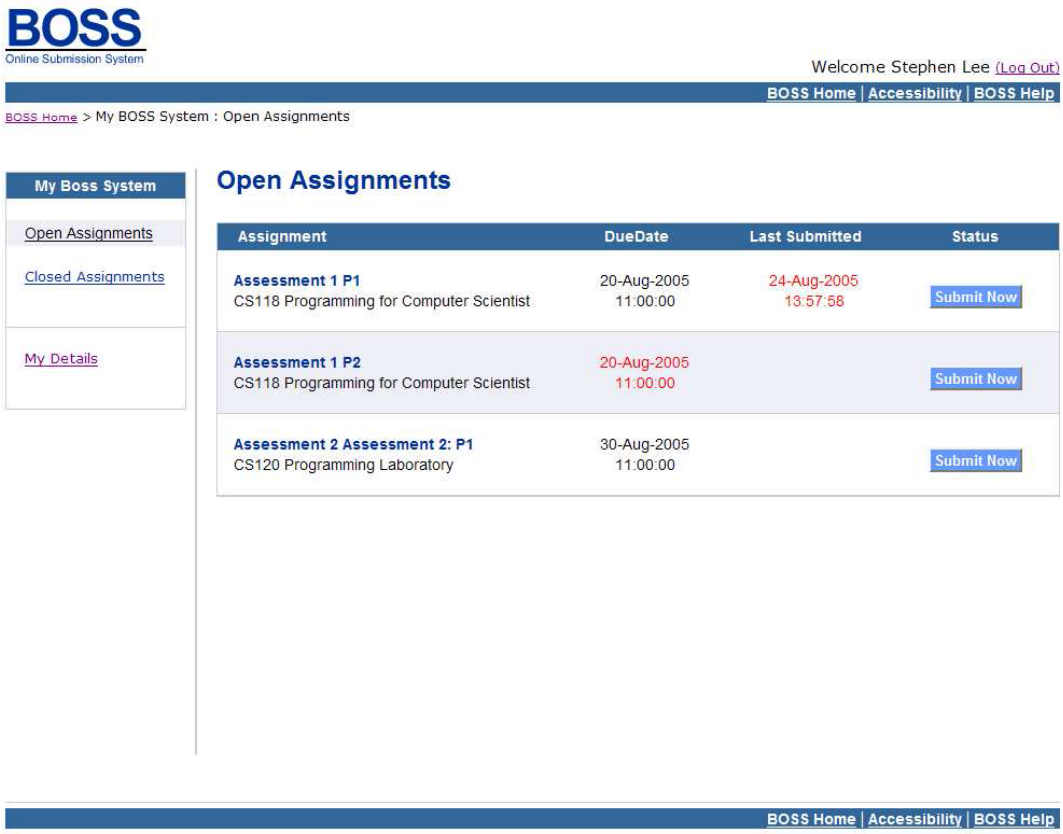


Figura 5.3: Interface do aluno

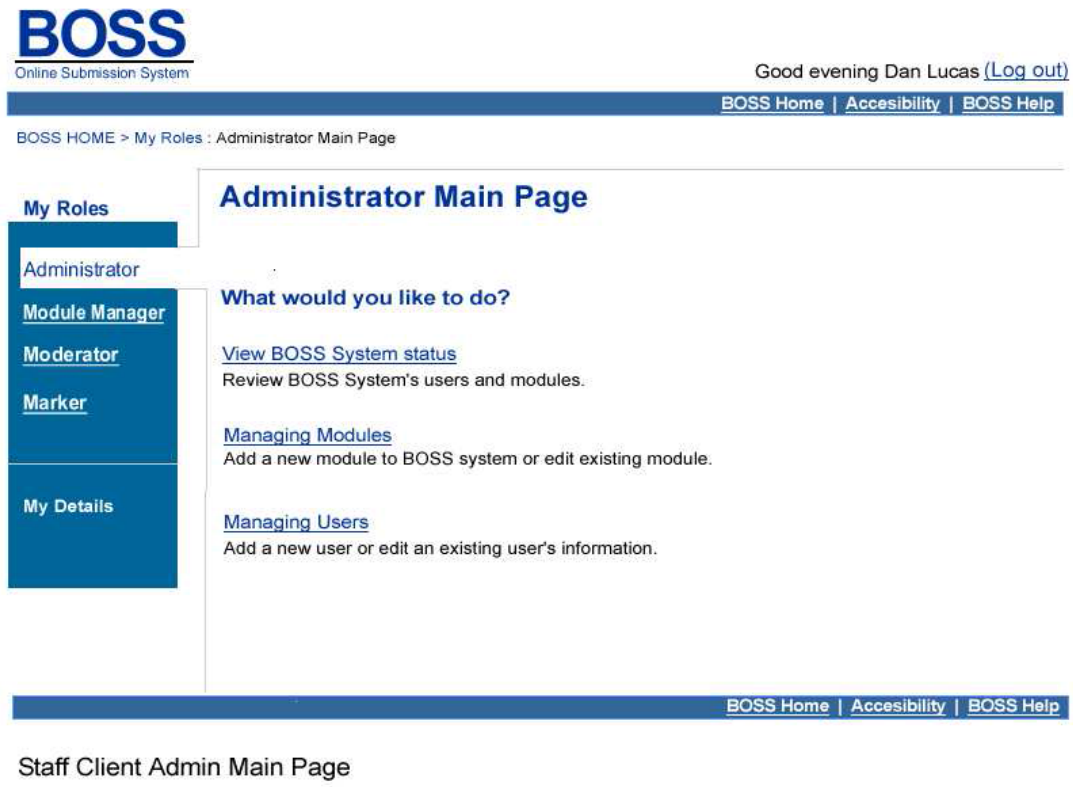


Figura 5.4: Interface de docente

As ajudas na navegação incluem uma barra de navegação na lateral esquerda e duas barras azuis (parte inferior e superior da janela) que permitem executar várias funções importantes para os utilizadores.

Os links para uma página de ajuda e acessibilidade estão posicionadas na barra azul no canto superior e inferior da janela. Além da navegação mencionada existem várias pequenas ferramentas de navegação secundaria que são úteis na exibição de mais informação, não ocupando demasiado espaço na janela. Toda a informação mais relevante é colocada na parte superior da página para quando uma página for carregada, o utilizador ser capaz de identificar facilmente a página onde se encontra. Após a submissão de uma solução, é gerada uma página com a informação que a sua submissão foi bem sucedida e que foi copiada para o servidor, como mostra a Figura 5.5; Note-se que esta notificação não contem ainda nenhum *feedback* sobre a correção do programa.

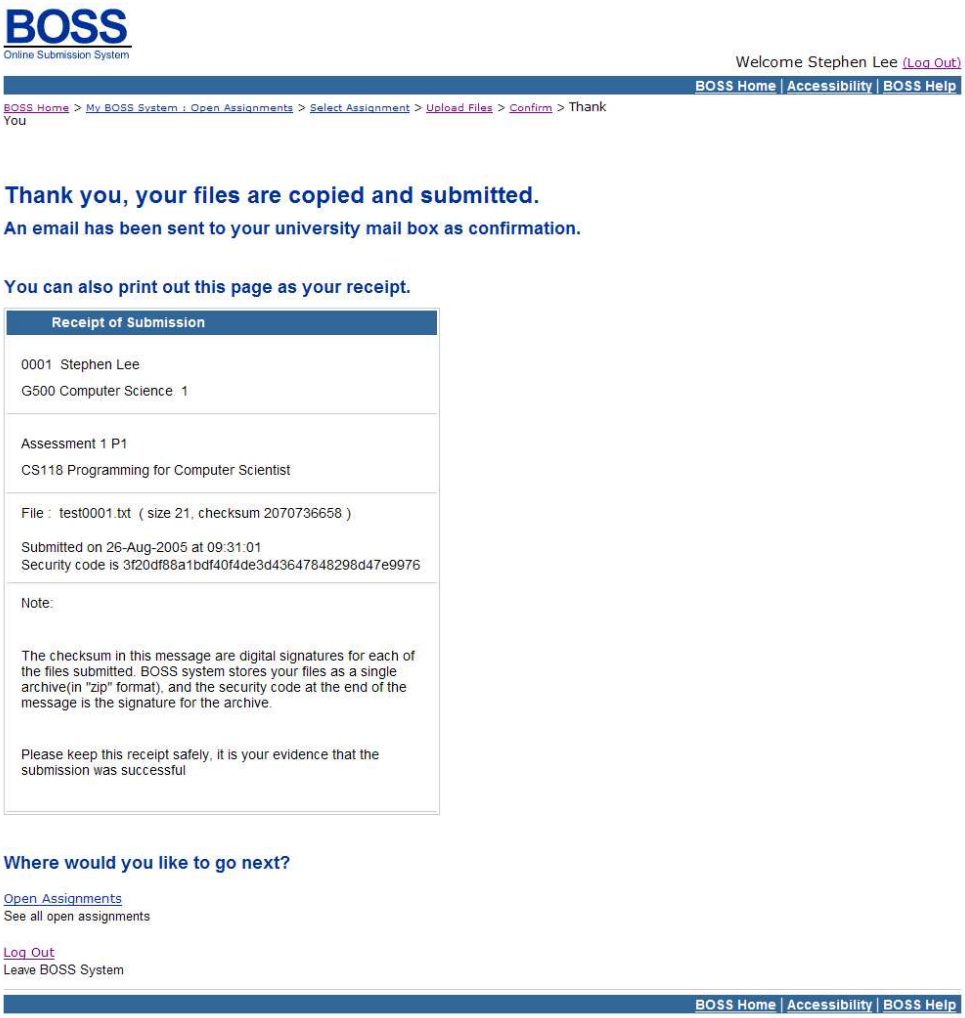


Figura 5.5: Interface da página de configuração de submissão

Com base na documentação consultada é possível dizer que o Boss testa os programas submetidos comparando os resultados realmente produzidos com os resultados esperados (fornecidos pelo professor) de forma a dar como *feedback* a indicação se o programa está correto ou errado. A equipa docente usa essa informação para classificar o desempenho do aluno nesse trabalho (formado por um ou mais exercícios de programação), havendo uma ou mais formas disponíveis para apresentar a classificação final. Pode-se então concluir que o Boss não faz uma análise do código submetido, avaliando apenas a sua correção.

5.1.2 Sistema Mooshak

Mooshak[28] (Leal et al., 2008; Leal et al., 2003), é uma aplicação Web criada para gerir concursos de programação, contudo sofreu algumas alterações para poder ser utilizado em ambiente de sala de aula. Num concurso de programação/ambiente pedagógico os exercícios são submetidos ao sistema e este faz uma correção automática e indica sucintamente se o programa é aceite como correto ou se não é aceite, mostrando então o tipo de erro cometido, conforme se detalha à frente. Em contexto pedagógico, tem sido usado em salas de aula como auxiliar de diagnóstico de funcionamento de programas estando também a ser usado na vertente competitiva.

O Mooshak apresenta diversas vistas consoante os requisitos e permissões de acesso aos dados geridos pelo sistema. O sistema inclui quatro vistas principais organizadas da seguinte forma:

- Vista do Administrador: permite aos diretores do concurso criar, gerir concursos equipas e utilizadores;
- Vista do Júri: que permite ao júri fornecer *feedback* aos concorrentes; validar, classificar / reavaliar programas; responder, colocar questões, controlar impressões e ver submissões resultados da avaliação (correções) e impressões;
- Vista do Concorrente: que permite aos concorrentes comunicar com os membros do júri, colocar perguntas, submeter programas e requerer impressões;

[28] http://academy.dei.uc.pt (versão 1)

- Vista do Público: que permite a qualquer utilizador da Internet seguir on-line o progresso do concurso.

Após o registo, o utilizador recebe um email com os dados de acesso (username e password). A partir daí, poderá ligar-se na máquina e começar a resolver problemas. Depois de fazer o login, o ecrã que aparece é algo como a Figura 5.6.

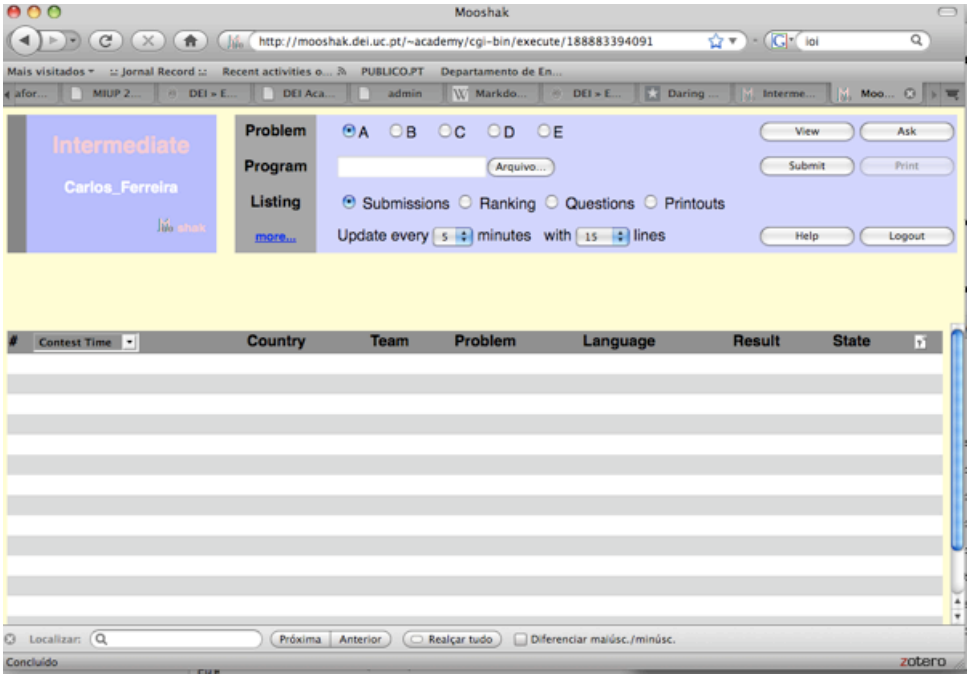


Figura 5.6: Interface da página de uma das vistas do concorrente

O concorrente pode então escolher um problema, clicando no botão correspondente. Se carregar no A, por exemplo, terá o seguinte problema (Figura 5.7):

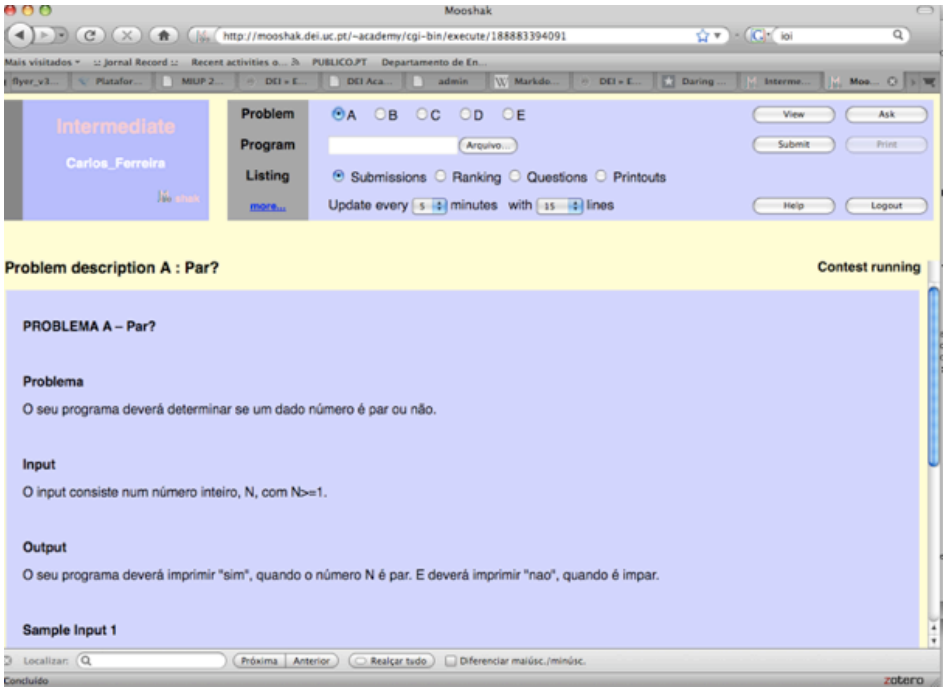


Figura 5.7: Interface da página da escolha do problema a resolver

O utilizador pode então submeter a solução com o botão "Submit". Após a submissão obtem uma resposta relativa à correcção do seu programa dentro de um conjunto prédefinido de possíveis de diagnósticos (Figura 5.8).

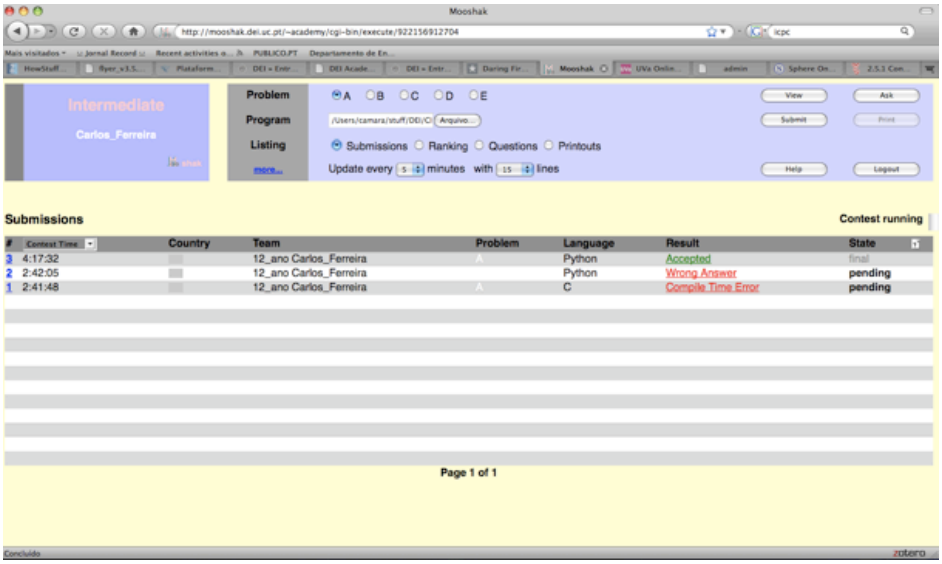


Figura 5.8: Interface da página da escolha do problema a resolver

O Mooshak permite que o professor lá coloque um grande número de problemas para serem resolvidos. De cada vez que cria um problema novo, o que é feito é colocar um enunciado e

um conjunto de casos de teste. Cada caso de teste consiste em dados de entrada e dados de saída - para um programa ser considerado correto terá de produzir os a saída correta em função dos dados de entrada para todos os testes incluídos no conjunto fornecido pelo autor do problema.

Após a validação do programa , o sistema original produz um resultado, que pode ser:

- Accepted – Tudo correto
- Presentation Error – Falta/sobram espaços
- Wrong Answer – Submissão errada. Há diferenças entre a saída produzida pelo programa do aluno e a saída esperada pelo menos em 1 caso de teste.
- Time Limit Exceeded – Programa excede limite de tempo.
- Runtime Error – Programa lança uma exceção. Ou seja, pára a meio da execução.
- Memory Limit Exceeded - Programa excede limite de memória.
- Invalid Function – Programa utiliza uma função inválida.

Contudo, existe atualmente uma versão orientada para o apoio ao ensino que, nas palavras do autor, "*já tem algum suporte para feedback. Sendo um sistema agnóstico relativamente à linguagem e com a avaliação baseada em casos de teste, não é de estranhar que o feedback se baseie na anotação dos testes. Por exemplo, os testes podem ser marcados para poderem ser mostrados (input e output), e podem ser associadas dicas a testes, que são mostrados caso estes falhem. Este feedback é incremental, sendo mostrado primeiro o feedback mais geral e, no caso de não haver progresso, é mostrado feedback mais específico.*" (Leal et al., 2010).

Os alunos podem submeter as suas soluções em qualquer momento, introduzindo um determinado requisito para poderem receber a avaliação.

Usando o Mooshak, algumas vantagens podem ser enumeradas:

- O alunos têm de se esforçar para compreender as matérias leccionadas nas aulas de algoritmos e estruturas de dados;
- Os alunos desenvolvem a sua aptidão de programar e implementam num número razoável de problemas;
- Os alunos aprendem a trabalhar em equipe;

- Os alunos vão acompanhando o seu progresso, enviando as suas soluções e recebendo *feedback* imediato;
- O alunos aprendem a pensar nos testes necessários para descobrir os seus erros e as respetivas soluções;
- Os professores podem automatizar a avaliação, e assim lidar com turmas maiores (Leal et al., 2008).

Em suma, o grande objectivo desta ferramenta é gerir os concursos tais como o SWERC (Southwestern Europe ACM Programming Contest), o MIUP (Maratona Inter-Universitária de Programação) e o ONI (Olimpíadas Nacionais de Informática). Porém, tem vindo também a ser usado em contexto pedagógico, nas salas de aulas como auxiliar de diagnóstico de funcionamento de programas. O sistema está a ser usado com sucesso em várias universidades, quer na vertente competitiva, quer na vertente pedagógica.

O sistema está disponível para instalação a partir da web. Além das funcionalidades associadas à avaliação automática de programas, gestão de concursos ou atividades pedagógicas, dispõe de um subsistema de ajuda que pode ser adaptado para Português, Inglês e Espanhol.

5.1.3 Sistema EduJudge

EduJudge[29] (Verdú et al., 2011; Castro et al.,...) é um projeto que envolve uma ferramenta de aprendizagem da programação ("UVA On-line Judge") existente on-line com um número significativo de problemas e com um ambiente educacional eficaz suportado pela plataforma Moodle e a ferramenta QUESTOURnamen[30] (Verdú et al., 2006), que permite aos professores configurar ambientes de trabalho individuais e em grupo com um conjunto de desafios a serem resolvidos pelos alunos em tempo definido pelo autor, sendo usado em ambientes de aprendizagem competitiva e colaborativa.

[29] http://www.edujudge.eu

30 The QUESTOURnament foi desenvolvida por um grupo de investigadores: Infraestructuras, Tecnologías, Aplicaciones y Servicios de Telecomunicaciones (ITAST) - http://www.itnt.uva.es/ and http://itastserver.tel.uva.es/formacion/.

Este sistema permite aos professores aplicar abordagens pedagógicas diferentes utilizando uma plataforma de e-learning, tornando os problemas fáceis de pesquisar através de um motor de busca e fornecendo uma avaliação automática das soluções apresentadas para os problemas. UVA On-line Judge é um sistema AAP on-line criado em 1995 com o objetivo de treinar os utilizadores para competições de programação em todo o mundo. Este sistema é muito interessante não apenas para os alunos, mas também para os professores.

Na sequência da criação e disponibilização do UVA On-line Judge, o projeto EduJudge surge para satisfazer a necessidade de uma ferramenta com caráter pedagógico mais acentuado, sobretudo para efeitos de avaliação e, desta forma, facilitar a sua utilização como uma atividade regular em cursos oficiais. Este é um projeto financiado com o apoio da Comissão Europeia, cujo objetivo é integrar o UVA On-line Judge num ambiente educacional eficaz: o Moodle Learning Management System e a ferramenta QUESTOURnament. Assim sendo, o sistema EduJudge pode ser adaptado a diferentes ambientes de ensino de programação e não apenas usado em avaliações para competições de alto nível.

EduJudge é composto por três subsistemas principais: um servidor de avaliação, um repositório de objetos de aprendizagem e uma interface do utilizador. A interação entre os diferentes componentes deste sistema está representada na Figura 5.9.

O servidor de avaliação é capaz de fornecer uma avaliação diferenciada (em vez de uma avaliação correto ou incorreto) e avaliar os diferentes tipos de problemas de programação: problemas com um único caso de teste, problemas com vários casos de teste, e problemas interativos em que a solução deve interagir com outro programa. Tal como o Mooshak, este ambiente também suporta uma grande variedade de linguagens de programação como Pascal, Java, C e C++.

O repositório melhora a acessibilidade a várias classes de problemas. O seu objetivo é armazenar os problemas de programação como Objetos de Aprendizagem (LOs, Learning Objects) e fornecê-los para o Sistema de Gestão de Aprendizagem e para o servidor de avaliação de uma maneira compatível. Atualmente e tal como já referido acima, este repositório contém uma grande variedade de problemas de programação que podem ser pesquisados por palavra-chave, autor, tipo, grau de dificuldade e da linguagem.

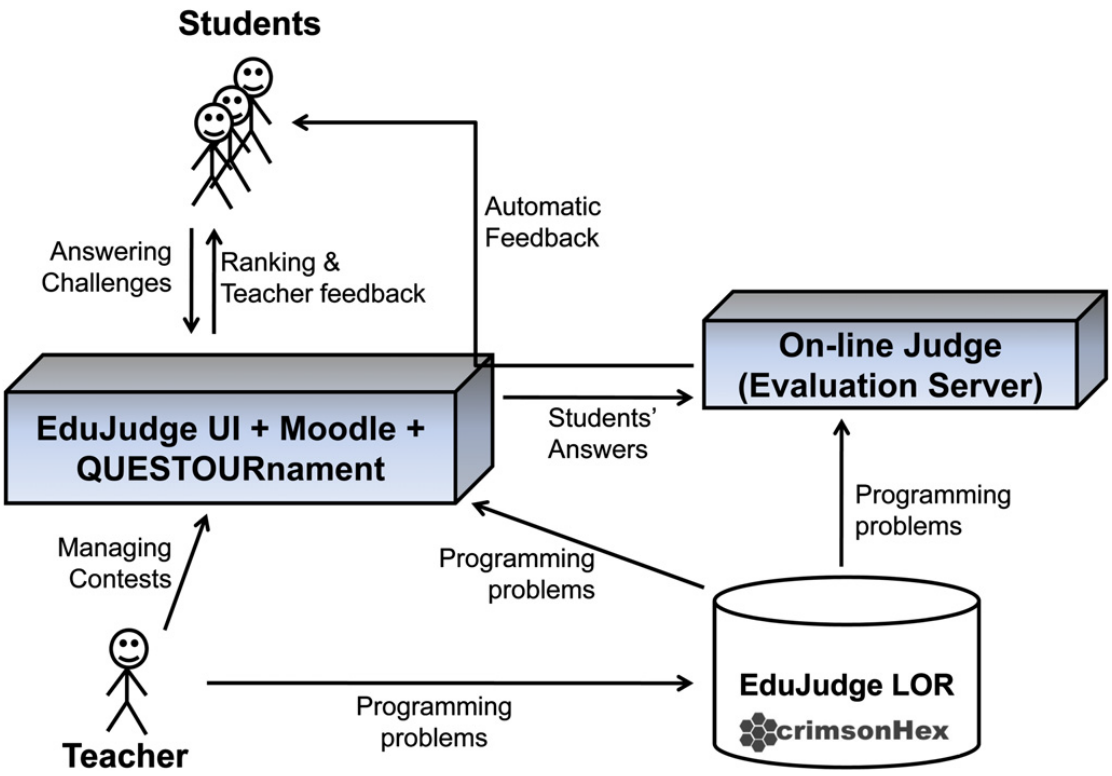


Figura 5.9: Estrutura e interações do sistema EduJudge (Verdú et al., 2011)

A interface do EduJudge consiste num conjunto de plugins e módulos para o Moodle, apoiando diferentes técnicas pedagógicas, em combinação com a avaliação automática de exercícios de programação recuperados do repositório de problemas.

Estão enumerados a seguir todos os passos necessários para criar um concurso de programação com o EduJudge num ambiente educacional:

1. O professor cria uma atividade no QUESTOURnament e adiciona desafios para o concurso. Cada desafio pode ser composto por um ou vários problemas de programação;
2. Os alunos devem responder aos desafios e obter *feedback* automático a partir do servidor de avaliação (On-line Judge);
3. A competição prossegue: o professor supervisiona todo o processo e os alunos verificam a sua classificação sempre com o *feedback* do professor.

Tanto os professores como os alunos podem usar EduJudge, com o fim de aumentar a sua motivação e desempenho. Os professores podem reutilizar, partilhar problemas e organizar concursos de programação educativos. Os alunos podem ter acesso a um número importante de problemas de programação de acordo com seu perfil e suas necessidades de aprendizagem, apresentar os seus programas como soluções e receber automaticamente o *feedback* do sistema. Os professores podem ainda acrescentar comentários ou alterar a classificação dada pelo sistema.

O sistema EduJudge fornece:

- Programação de problemas com uma estrutura adequada, em que as soluções podem ser avaliadas automaticamente por um servidor on-line;
- Um motor de busca para que os problemas sejam mais facilmente acessíveis para os utilizadores;
- Uma oportunidade para aplicar diferentes abordagens pedagógicas que utilizam o e-learning como a plataforma Moodle.

Tanto quanto foi possível apurar, o sistema EduJudge, baseado no UVA On-line Junge, após avaliar a solução submetida pelo aluno devolve-lhe um *feedback* que apenas indica se o resultado produzido está correto (passa em todos os testes) ou não (falha pelo menos em um). Com base nessa avaliação e na orientação do professor (que define os critérios de pontuação), as respostas são pontuadas e mostradas ao aluno para incentivo à participação. O sistema mostra permanentemente a classificação resumida atual que poderá ser individual ou por equipa.

Resultados de estudos feitos pelos autores, indicam que a utilização do sistema EduJudge tem efeitos importantes sobre os resultados escolares dos alunos. Os alunos que usaram este sistema obtiveram melhores notas finais (Verdú et al., 2011). Portanto, este sistema pode apoiar estratégias de ensino eficazes para a aprendizagem de programação baseada na competição e avaliação automática.

5.1.4 Sistema Codeboard

O Codeboard[31] é um IDE baseado na web para ensinar a programar apoiando os professores em sala de aula ou fora dela. É uma ferramenta poderosa mas de simples utilização que permite aos professores oferecer aulas mais envolventes e interessantes. Um dos principais objetivos é a análise do código submetido pelos alunos (o sistema suporta variadas linguagens de programação, à semelhança das ferramentas anteriores) com vista a fornecer *feedback* imediato sobre a correcção desse código, para ajuda dos alunos.

- Nas MOOCs (Massive Open Online Courses), que são plataformas muito usadas e que permitem guardar vários conteúdos para o ensino, mas que não auxiliam os professores na criação de exercícios de programação e não permitem o *feedback* instantâneo. O Codeboard pode colmatar esse problema, permitindo que os professores forneçam exercícios em que os alunos não têm que instalar qualquer *software* recebendo o *feedback* do seu trabalho.
- Em sala de aula, depois do professor introduzir um novo conceito, deve fornecer sempre exercícios para o enfatizar; disponibilizando esses exercícios via Codeboard pode, posteriormente, ter acesso aos exercícios feitos pelos alunos, mostrando a solução de um dos alunos para toda a turma (sem problemas de ter que mudar de computador).
- Fora da sala de aula, os alunos podem compilar e submeter as suas soluções no Codeboard e este torna a tarefa do professor muito mais simples na medida em que o sistema verifica automaticamente e classifica sendo estes resultados facilmente acessíveis por ambas as partes.

Um projeto em Codeboard é composto por várias componentes: nome do projeto, descrição do projeto, a linguagem de programação utilizada, acessibilidade, lista de proprietários (professores) e lista de utilizadores (alunos), permitindo submissões e configurações. A Figura 5.10, ilustra a página de entrada no sistema, destacando-se as diferentes funcionalidades disponíveis.

[31] https://codeboard.io/about

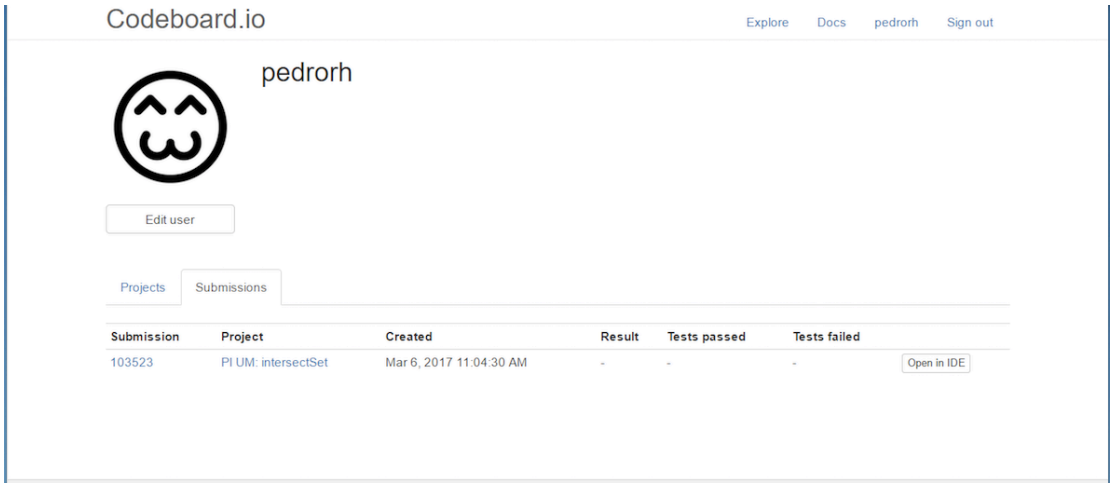


Figura 5.10: Página de entrada no Sistema Codeboard

Todos os projetos possuem um ficheiro de configuração denominado "codeboard.json" que define para cada linguagem de programação como é o que o projeto é compilado ou executado. Podem ser visualizadas caixas de texto com o código (total ou parcialmente escrito pelos alunos) para a compilação e execução dos projetos, como se vê na Figura 5.11. Os programas em execução são interrompidos se alguns dos seguintes limites for ultrapassado (em termos de segurança):

- Tempo total de CPU usado pelo projeto;
- Tempo total de sessão para executar um projeto;
- Número de threads criados por um projeto;
- Número de caracteres de saída criados por um projeto.

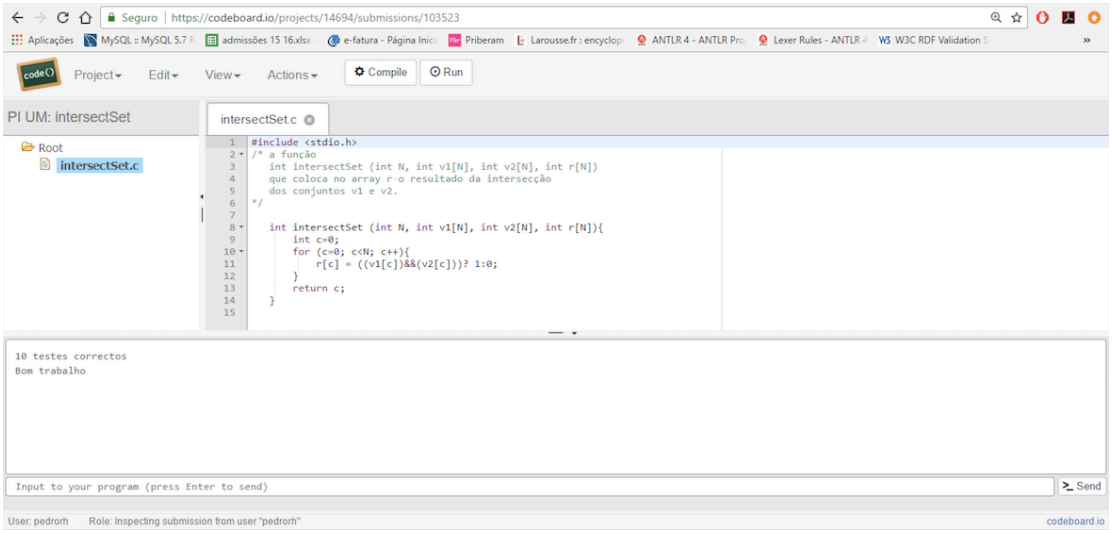


Figura 5.11: Interface com a visualização do código a compilar

Para a avaliação, dois tipos de classificação são suportadas:

- Classificação automática, usando uma sequência de resultados. Os projetos no Codeboard podem implementar a classificação automática simplesmente imprimindo uma string especial como a última saída quando o projeto é executado. Essa string deve ser da forma:

`<!–@test=the_grade_value; num_test_passed; num_test_failed;–>` (na submissão, o Codeboard compila e executa o programa apresentado e verifica se a saída do programa termina com uma string que satisfaça o formato da sequência de resultados.)

- Classificação automática, usando testes unitários. O Codeboard suporta testes unitários para o seguinte linguagens de programação:
  - Java-JUnit
  - Haskell-HSpec
  - Python-UnitTest

  Na submissão, o Codeboard executará todos os testes unitários disponíveis na pasta de teste.

As Figuras 5.12 e 5.13 ilustram situações em que a solução do aluno (errada) não passou a todos os testes.

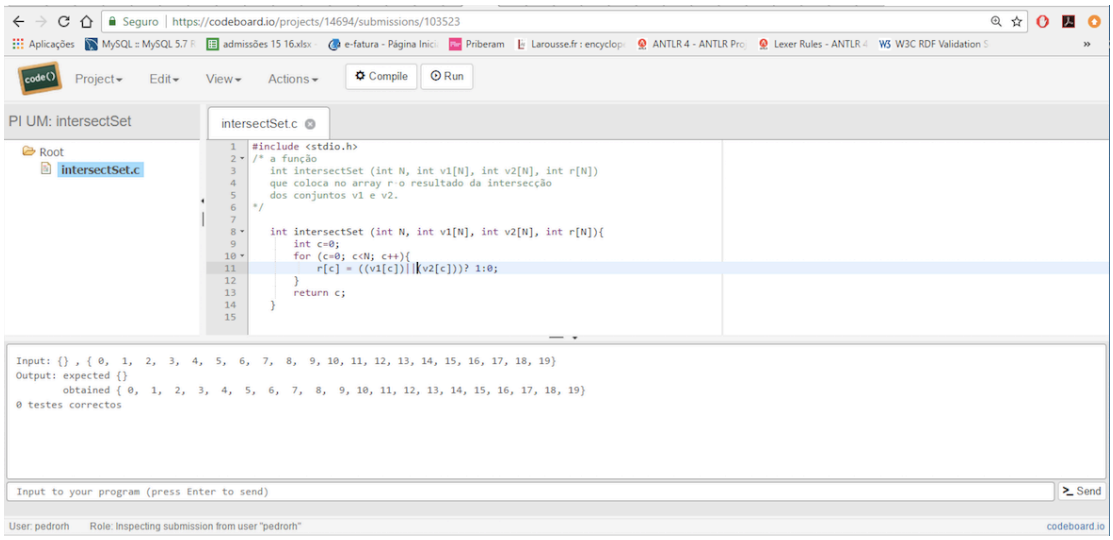


Figura 5.12: Visualização de uma solução errada do aluno (input e o output)

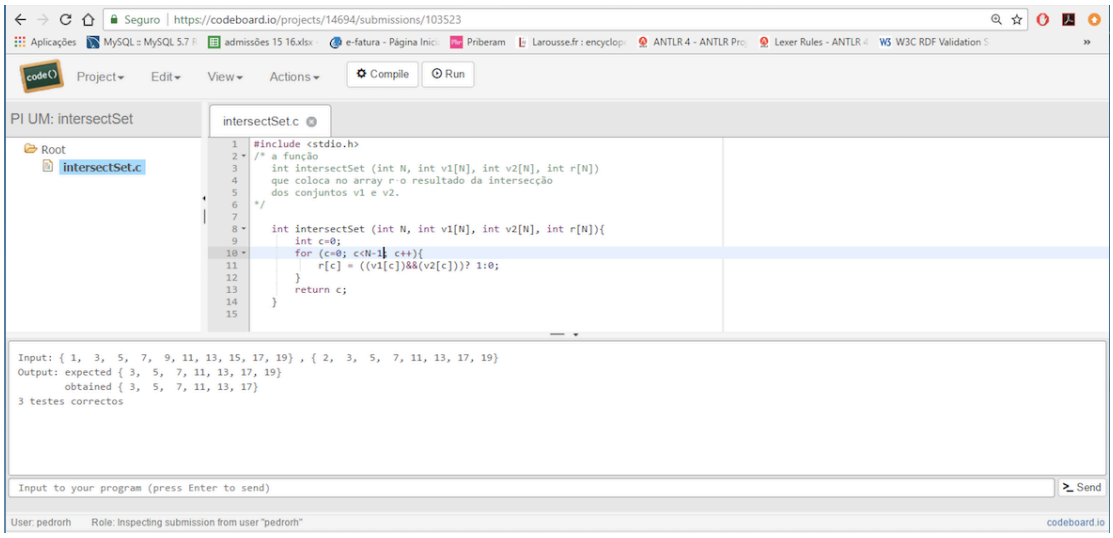


Figura 5.13: Visualização de uma solução errada de um aluno (input e output)

O Codeboard pode ser integrado em qualquer plataforma que suporte ao ensino, LMS, visto que assegura a interoperabilidade LTI (Learning Tools Interoperability) que permite que as plataformas educacionais se integrem de forma segura com ferramentas alocadas externamente. Uma plataforma educacional pode enviar informações do aluno para o Codeboard e, em troca, o Codeboard reporta as notas dos alunos de volta (Sankpal, ?).

Na Listagem seguinte mostra-se um fragmento de uma folha de exercícios de uma disciplina universitária de introdução ao C, onde a equipa docente à frente do enunciado coloca um link direto para a página específica do Codeboard que está pronta a receber a resolução do exercício em causa e a permitir o seu teste imediato. Para efeitos de ilustração, o exercício abaixo listado corresponde ao que é testado correto ou incorretamente nas Figuras 5.11, 5.12

Exemplo de um exercício da aula:

Uma forma de representar conjuntos de índices consiste em usar um array de inteiros contendo 1 ou 0 consoante esse índice pertença ou não ao conjunto. Assim o conjunto {1; 4; 7} seria representado por um array em que as primeiras 8 posições conteriam {0,1,0,0,1,0,0,1}.
Apresente uma definição da função
int intersectSet (int N, int v1[N], int v2[N], int r[N])
que coloca no array r o resultado da interseção dos conjuntos v1 e v2

(https://codeboard.io/projects/14694).

A integração é muito fácil e, portanto, é utilizada por vários cursos em plataformas educativas para ensinar programação (Sankpal, ?). Esta faceta juntamente com o facto de uso muito simples e leve, fazem do Codeboard uma das plataformas mais promissoras nesta área da avaliação automática para retorno imediato de *feedback* dentro ou fora de aula.

5.2 Sumário

Para os professores, os sistemas de avaliação automática de programas (AAP) devem ajudar a automatizar a maior parte do trabalho de avaliação. Isso incluí aspectos como ser fácil criar testes automáticos e automatizar a avaliação das soluções desenvolvidas pelos alunos e sua publicação.

Mas o mais importante, no âmbito do processo de ensino aprendizagem, é que um sistema AAP ajuda os alunos aumentando-lhes a auto-confiança e motivação na medida em que fornece *feedback* instantâneo sobre a solução apresentada, permitindo que os alunos possam trabalhar ao seu próprio ritmo. Esse *feedback* resulta de avaliar o comportamento da dita solução procedendo automaticamente à execução de uma série de testes de modo a apurar aqueles que passaram e aqueles que falharam e, preferencialmente a identificar ainda as razões das falhas. Esta interatividade com o aluno tenta envolver os alunos no curso e assim

ajudar a melhorar a aprendizagem e, consequentemente, reduzir as taxas de insucesso dos cursos.

Tirando partido do espírito humano de competição, a aprendizagem competitiva aumenta o compromisso e leva a um maior envolvimento dos alunos em atividades práticas. Então, competições com avaliação automática estão a tornar-se muito importantes para a prática de programação. A aprendizagem competitiva leva a um maior envolvimento dos alunos. No entanto, as diferenças de motivação e de sentimentos entre vencidos e vencedores podem existir, podendo ser atenuadas através de diferentes práticas, como seja, focar a atividade essencialmente na aprendizagem e diversão.

A programação tem sido tradicionalmente ensinada e praticada como uma atividade fundamentalmente individual. No entanto, ao longo dos últimos anos, foram adotadas diferentes práticas de aprendizagem colaborativa como a programação em pares e os projetos de equipe. Assim, os alunos aumentam a sua auto-confiança, produzem melhores programas e melhoram seu desempenho na programação.

Na verdade, uma vez que nem todos os alunos são motivados pelas mesmas coisas, é interessante e importante ser capaz de definir diferentes ambientes de aprendizagem: colaborativo, competitivo ou misto.

O papel do trabalho em grupo na educação e o ambiente colaborativo entre estudantes é discutido em (Boas et al., 2013; Fonte et al., 2014). Neste trabalho, os autores propõem o uso de técnicas de integração contínua, habituais em *Agile Development* (Elliott et al., 2015) (Awad, 2005), para apoiar o trabalho do grupo fornecendo *feedback* imediato aos alunos e professores.

6. PROPOSTA: COMBINANDO ANIMAÇÃO E AVALIAÇÃO AUTOMÁTICA

Considerando tudo o que foi mencionado em capítulos anteriores, a razão principal deste trabalho de doutoramento reside na dificuldade do processo ensino/aprendizagem da programação e no insucesso escolar que daí deriva. Assim e após a investigação de um conjunto de trabalhos científicos relativos ao insucesso da programação e contributos para a sua resolução, vão-se apresentar neste capítulo propostas de solução, com o intuito de contribuir para minorar este problema.

Como descrito anteriormente, o principal objetivo é:

- fornecer os meios para uma compreensão mais fácil de programas, e assim ajudar na escrita de novos;
- fazer com que os alunos aumentem a sua capacidade de praticar regularmente a programação, desde o primeiro dia, obtendo *feedback* imediato.

Para tal, estudos experimentais, no âmbito escolar, serão essenciais para o desenvolvimento ou uso de plataformas de ajuda à aprendizagem. Estas ferramentas serão elementos de apoio ao ensino para o aumento da capacidade dos alunos de resolução do seu problema. É importante auxiliar os alunos na transição dos conhecimentos básicos, para uma visão de uma solução algorítmica, daí a importância destas ferramentas para apoiar este processo. O objetivo é fazer com que os alunos aumentem a sua capacidade de praticar com regularidade a programação, desde o primeiro dia, pois acreditamos que desta forma o seu sucesso escolar vai aumentar.

Para isso, como já referido atrás, são usadas ferramentas que permitem a visualização de muitos dos conceitos (estáticos e dinâmicos) envolvidos na programação; por causa das maiores dificuldades dos alunos estarem ligados à forma como o programa funciona, as ferramentas em causa não se limitam a mostrar através de ícons apropriados as componentes de um dado programa, mas recorrem à sua animação para auxiliar na compreensão dessas mesmas componentes. Uma abordagem deste tipo pode ser motivadora, os níveis de dificuldade podem ser graduais, permitindo a todos os alunos aprender a programar com mais facilidade ou melhor. Além disso, estas ferramentas de visualização e animação, permitem ao professor usar uma abordagem mais interativa e experimental e menos expositiva. Outro ponto importante a referir desta abordagem consiste em centrar-se primeiro na resolução de problemas de programação em vez das características sintáticas da linguagem de programação.

A existência e uso de vários sistemas e ferramentas que auxiliam o aluno na fase de planeamento e criação do algoritmo é muito importante para que, na elaboração dos seus programas, os problemas de conceção sejam minimizados. Porém, e como já referido não foi possível encontrar uma consistência de resultados na melhoria da aprendizagem com o uso de ferramentas de visualização e animação. Daí a convicção de que estas ferramentas requerem a junção de outras abordagens que melhor estimulem a auto-confiança.

Como receber *feedback* imediato é essencial para a aquisição de conhecimento, ferramentas que permitam a submissão de programas desenvolvidos pelos alunos devem ser incorporadas nas atividades letivas, permitindo-lhes assim testar os seus trabalhos recebendo logo uma

avaliação daquilo que desenvolveram; desta forma os sistemas de avaliação automática podem melhorar significativamente o desempenho dos alunos.

Com o objetivo de aumentar a motivação e a autoconfiança dos alunos dos cursos de Introdução à Programação, apresenta-se neste capítulo uma proposta que visa sugerir diferentes abordagens, suportadas em recursos informáticos, para atingir esse objetivo.

6.1 Abordagens Propostas

Cabe ao docente, perceber as fragilidades dos alunos no processo de aprendizagem e resolve-las rapidamente. Para atingir os objetivos e com os recursos computacionais disponíveis atualmente, é proposto nesta tese combinar técnicas de animação e avaliação automática.

Em particular, são propostas duas técnicas concebidas para apoiar o ensino da programação: uma, mais antiga, a Animação de Programas que pretende tirar partido da nossa acuidade visual e do efeito da simulação para facilitar a compreensão dos programas e algoritmos subjacentes; e outra, muito mais recente, que aposta no recurso a sistemas de Avaliação Automática de Programas para incentivar os alunos a continuar a praticar, providenciando-lhes *feedback* imediato após a conclusão da escrita de um programa. Com base na combinação destas técnicas e respetivas ferramentas, iremos enunciar duas possíveis abordagens:

6.6.1 Abordagem 1 - Avaliação seguida de Animação

O aluno começa por ter de resolver um dado problema, devendo submeter a solução à avaliação automática para receber o respetivo *feedback*; depois procede à análise da solução correta com recurso à animação.

Para concretizar esta abordagem, o professor prepara para cada tema a apresentar, um conjunto de N problemas relativos a esse tema de dificuldade semelhante. Para cada problema do conjunto P_i (i=1,2,...,N) procede seguindo os passos A1 a A4 para preparação da aula conforme ilustrado na Figura 6.1. É importante um enunciado claro para cada problema (A2). Para os sistemas de avaliação e de animação, deve preparar respetivamente os vetores de teste (A3) e uma resolução detalhada e comentada (A4).

Uma vez preparada a aula, esta consistirá em uma ou mais etapas em que cada etapa corresponde ao estudo de um problema P_i do conjunto definido. Cada etapa é formada pelos passos B1 a B2 conforme representado na Figura 6.2.

Para cada problema P_i do conjunto, pede que o aluno analise o enunciado, desenvolva o algoritmo e o codifique passando a testá-lo com o sistema de AA. Enquanto não se esgotar o tempo previsto pelo professor para este passo, o aluno pode ir iterando entre resolução, submissão e avaliação até à solução correta (B1). Ao fim desse tempo, o professor fornece a sua solução e pede ao aluno que a analise cuidadosamente recorrendo à ferramenta de animação (B2) procurando que assimilem o conhecimento daí derivado.

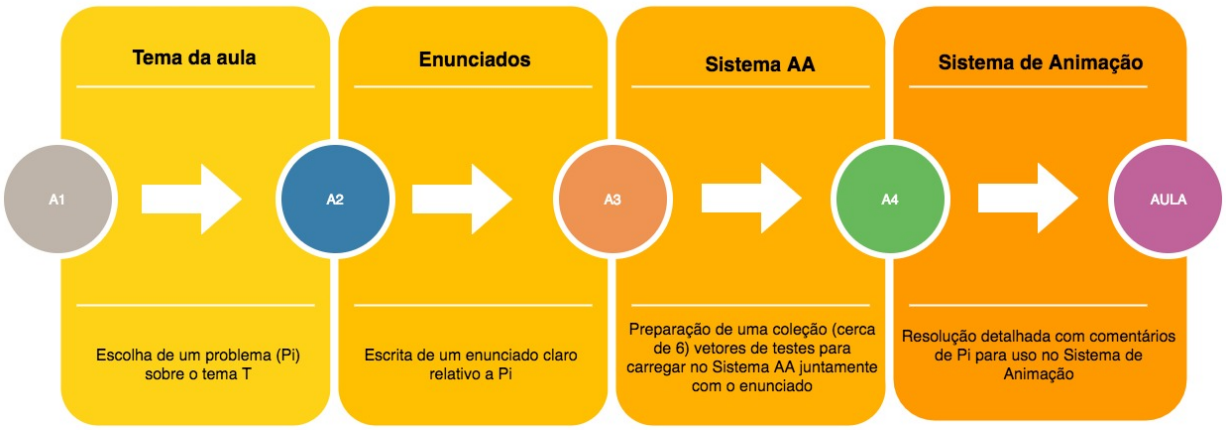


Figura 6.1: Representação da preparação da aula por parte do Professor para a abordagem 1

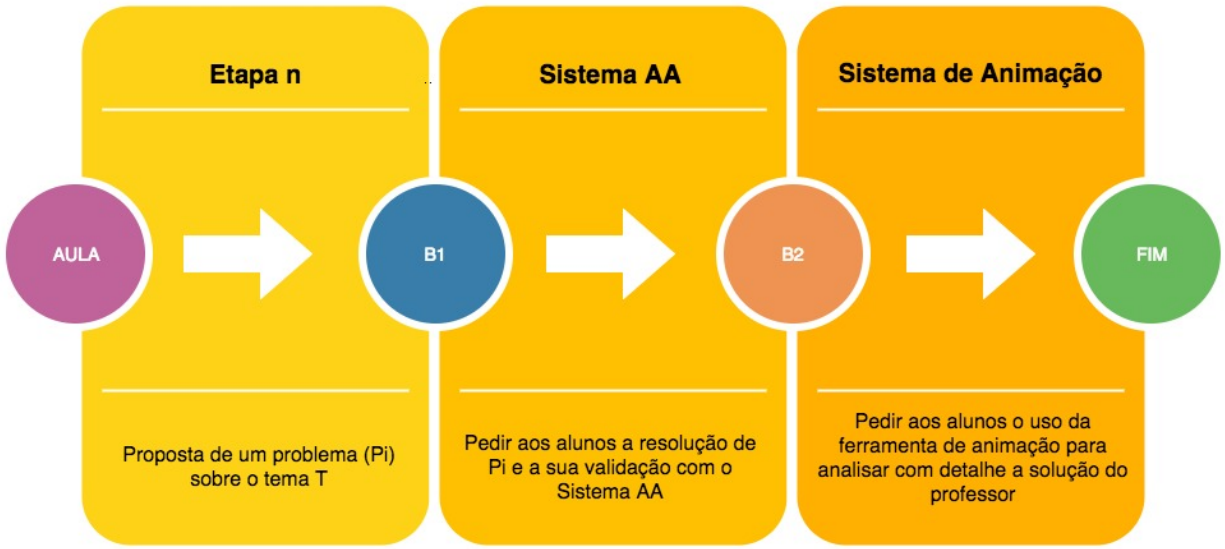


Figura 6.2: Representação dos passos do aluno para a aula da abordagem 1

6.1.2 Abordagem 2 - Animação seguida da Avaliação

Primeiro o aluno é exposto à análise dos problemas e sua resolução, com o apoio da Animação; Depois passa a uma segunda fase onde, o próprio aluno, resolve problemas semelhantes, recorrendo à avaliação automática para obter o respetivo *feedback* rápido de cada problema resolvido. Para cada exercício submetido o aluno poderá recorrer de novo à animação caso tenha falhado, para melhor entender os motivos de erro.

Por cada tema a ensinar, o professor prepara dois conjuntos, C_a e C_b, de exercícios análogos seguindo os passos A1 a A4 esquematizados na Figura 6.3.

A aula terá duas fases sequenciais B1 e B2 representadas na Figura 6.4. Para cada exercício P_i do primeiro conjunto C_a o professor discute o enunciado, esquematiza a resolução (esboça um algoritmo) e apresenta o programa que o resolve de modo a que o aluno possa fazer a sua animação e assim analisar/compreender a solução correta. Para cada exercício P_j do segundo conjunto C_b, após discutir o enunciado, o professor pede aos alunos que o resolvam e testem a solução obtida através de um sistema de avaliação automática. Enquanto não se esgotar o tempo previsto pelo professor para este passo, o aluno pode ir iterando entre resolução, submissão e avaliação até à solução correta. Num terceiro momento, o professor pode discutir com os alunos o *feedback* recebido do avaliador, podendo nessa altura o aluno reforçar a lição usando a animação mais uma vez para compreender a sua falha.

Figura 6.3: Representação da preparação da aula por parte do Professor para a abordagem 2

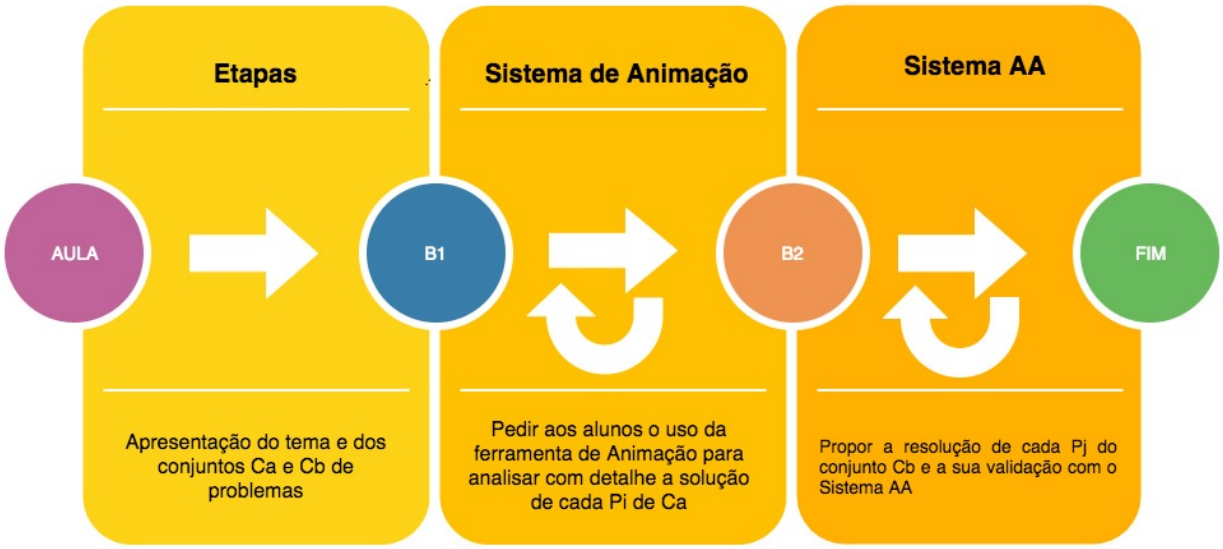


Figura 6.4: Representação dos passos do aluno para a aula da abordagem 2

6.2 Sumário

Face à identificação das dificuldades sentidas pelos alunos de Programação e com base nos estudos sobre Motivação e sobre sistemas Animação de Programas e de Avaliação Automática de Programas (sua génese, evolução das implementações e sucesso já obtido) propuseram-se neste capítulo duas abordagens pedagógicas alternativas que combinam essas duas técnicas com vista a tornar mais eficaz o processo de ensino aprendizagem da Programação. O objetivo de ambas é duplo: estimular a motivação aumentando o envolvimento e a participação, facilitando a compreensão e aumentar a confiança e auto-estima fazendo com que os alunos aumentem a sua capacidade de praticar com regularidade a programação, desde o primeiro dia. A estratégia aqui apresentada visa melhorar a atividade de ensino/aprendizagem em cursos introdutórios de programação combinando *feedback* imediato fornecido por sistemas automáticos de avaliação com ferramentas de animação.

A primeira abordagem é mais adequada para apoio a aulas puramente experimentais em sistema de ensino mais tradicionais em que os conceitos e a abordagem a cada tema é feita em aulas teóricas separadas; nesse contexto nas práticas laboratoriais apenas se pretende levar os alunos a exercitarem esses conhecimentos.

A segunda abordagem é mais adequada para apoio a aulas teórico-práticas em que os novos temas são expostos e o aluno é logo confrontado com a sua experimentação (ver soluções e tentar replicá-las) e é sobretudo mais conveniente para nela se basear um sistema de suporte ao auto-estudo.

Nos próximos capítulos as abordagens aqui suscintamente introduzidas serão aprofundadas e validadas. Nomeadamente serão discutidos experimentos realizados em sala de aula, sendo mostrados exemplos concretos. Um protótipo de um ambiente de apoio ao ensino baseado na 2ª abordagem será também apresentado.

7. EXPERIMENTOS EM SALA DE AULA

A proposta de trabalho que se fez, no Capítulo 6, tem como objetivo principal diminuir as dificuldades sentidas pelos alunos que se encontram numa fase de aprendizagem de programação. Para tal, estudos experimentais, no âmbito escolar, são essenciais para validar essas propostas, contribuindo naturalmente para as aperfeiçoar. Antes de passar à descrição e discussão dos experimentos conduzidos em sala de aula para aplicar as abordagens propostas, é importante relembrar que o objetivo dessas propostas é fazer com que os alunos aumentem a sua capacidade de praticar com regularidade a programação, desde o primeiro dia, fornecendo-lhes instrumentos que facilitem a compreensão dos programas no sentido de elevar a motivação, o envolvimento e o desempenho dos alunos.

7.1 Concepção e realização de um experimento para aferir a abordagem 1

Nesta secção será apresentada uma descrição do primeiro experimento realizado para aferir a abordagem 1.

Os principais objectivos para este primeiro experimento foram:

- compreender o comportamento dos alunos que enfrentam uma situação nova e diferente;
- observar se os alunos estão envolvidos e motivados;
- entender quais são as principais dificuldades dos alunos do primeiro ano, quando enfrentam uma tarefa de programação: a interpretação do enunciado, ou o desenvolvimento de algoritmos e sua codificação numa determinada linguagem.
- Verificar a eficácia da abordagem proposta

7.1.1 Planeamento do experimento

Para atingir os objectivos acima referidos, fazendo uso dos recursos de computação disponíveis e de acordo com a 1ª proposta metodológica do capítulo anterior, que sugere combinar técnicas de animação e avaliação automática, há um conjunto de passos a seguir pelo aluno e pelo docente:

- Passo 1: Para cada tópico a ensinar, o professor prepara três exercícios semelhantes;
- Passo 2: Para o primeiro exercício, o professor deve analisar com os alunos o enunciado do problema, e, em seguida, pedir-lhes para resolver o problema e testar a solução produzida usando o Sistema de Avaliação de Automático, AA, selecionado. O professor poderá, no fim de cada validação, discutir com cada aluno o feedback recebido do Sistema de AA;
- Passo 3: O professor fornece uma proposta de solução do exercício e o aluno deve utilizar o sistema de animação selecionado, para animar a execução do programa a fim de analisar com cuidado e entender a solução correta e seu comportamento;
- Passo 4: Repetir os passos 2 e 3 para os restantes exercícios, sobre o tema em estudo, escolhidos pelo professor para a aula.

Esta abordagem pressupõe que o professor selecione uma ferramenta de animação eficaz e fácil de usar, bem como escolha um avaliador automático que além de amigável retorne um feedback que forneça um diagnóstico e, se possível, comente a qualidade do código. As ferramentas escolhidas para os experimentos aqui relatados serão o Jeliot e o Mooshak, apresentados nos capítulos anteriores.

Para este experimento específico, escolheu-se ensinar um dos tópicos introdutórios da Programação "*sequências numéricas e estruturas de decisão e repetição*", tendo-se, então, desenvolvidos os três enunciados seguintes, com o cuidado de serem bastante semelhantes:

a) Construa um algoritmo que leia uma quantidade não determinada de números inteiros positivos. Calcule a quantidade de números pares e ímpares lidos, assim como a média (float) dos valores pares. O número que encerrará a leitura será o 0 (zero).
b) Dado um número 'm' e um número 'n', inteiros positivos, ler 'n' idades mostrando todas as idades maiores que 'm'. No final deve mostrar a média (float) das idades.
c) Dadas as temperaturas (float) de 6 dias do mês de Janeiro (valores entre -50° e 50°), mostre a temperatura máxima e mínima. Classifique também o mês como "frio" ou "quente" conforme teve mais dias com temperaturas negativas ou mais dias com temperaturas positivas (zero incluído). Em caso de igualdade considere que o mês é "frio".

Depois de decididas as ferramentas concretas a usar, o tema da aula experimental, e os exercícios para resolver, foi necessário escrever um plano cuidado da aula, para que todos os alunos conseguissem entender o que são convidados a fazer e como devem proceder. Para o efeito, foi esboçado um plano de aula com uma tarefa composta pelas seguintes etapas:

1) Entrar no sistema Mooshak (AES);
2) Escolher o exercício (A ou B ou C de acordo com a respetiva fase);
3) Ler atentamente o enunciado;
4) Desenvolver um algoritmo e codifica-lo em Java, a fim de resolver o exercício proposto (usar o NetBeans IDE);
5) Apresentar o programa escrito, para ser avaliada pelo Mooshak;
6) Analisar o resultado sobre a avaliação produzida pelo Mooshak;
7) Se o resultado não for "Accepted", voltar ao passo 4 e fazer as correções necessárias;
8) Se o resultado for "Accepted", passar para a visualização e animação do programa proposto pelo professor para resolver o problema com a ferramenta Jeliot;
9) Se o exercício atual é o terceiro, C, terminar o experimento; se não, voltar para o passo 2.

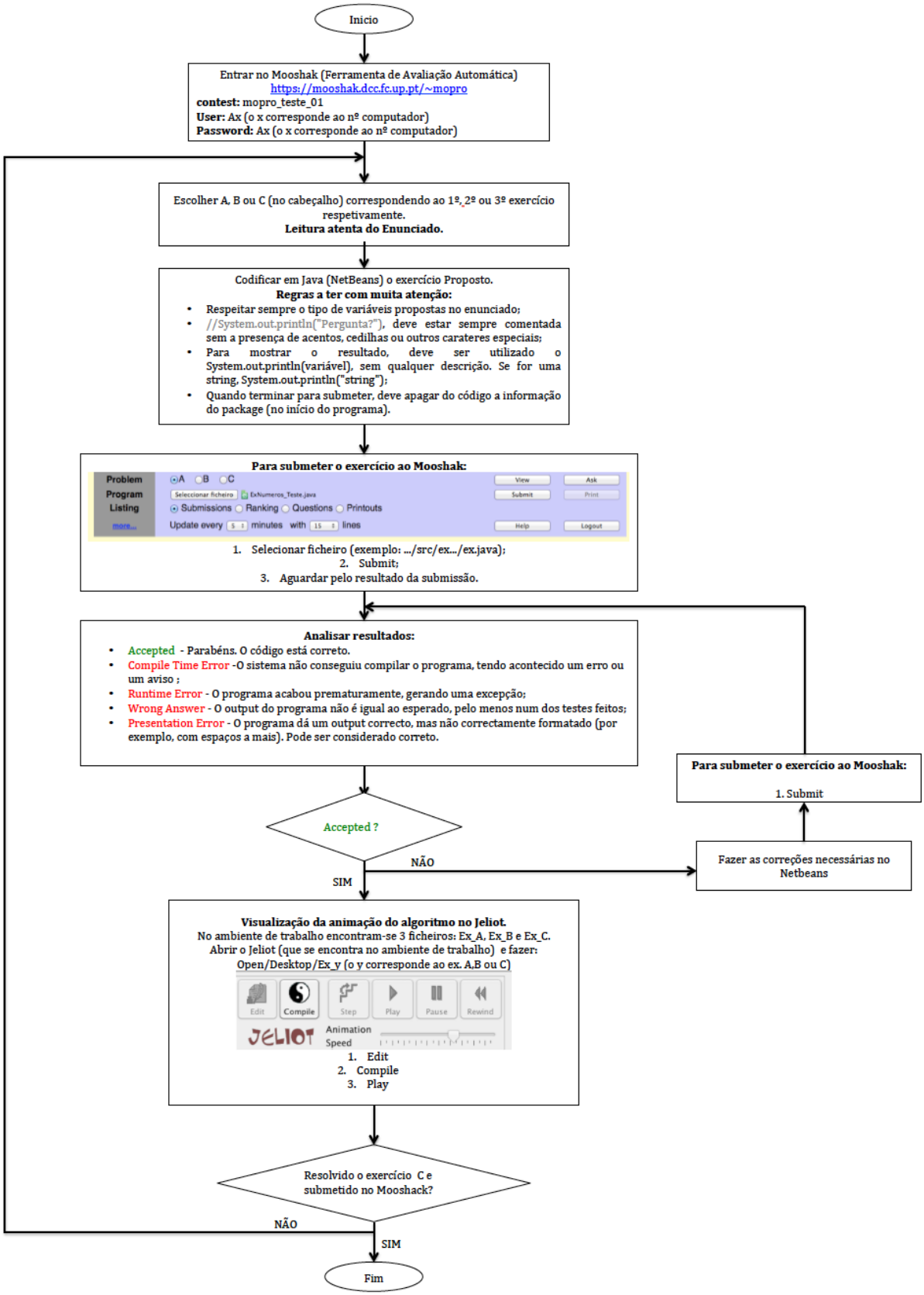


Figura 7.1: F Fluxograma representativo do experimento (abordagem 1): resolver três problemas em Java, obter feedback e visualizar a animação do algoritmo no Jeliot

Este plano foi desenhado como um fluxograma tendo sido impresso e distribuído a todos os alunos presentes na aula, como o apresentado na Figura 7.1.

Antes de iniciar a experiência, o Jelliot foi instalado em todos os computadores da sala de aula, e o Mooshak foi configurado e preparado num servidor fornecido pelo Departamento de Ciências de Computadores da FCUP, a fim de ser acedido pelos alunos via Internet (Mooshak é uma ferramenta acessível on-line).

7.1.2 Condução do experimento

O experimento foi feito com 28 alunos do 1º ano (num curso de Engenharia). Cada sessão foi supervisionada por dois professores que permaneceram todo o tempo na sala de aula para ajudar os alunos, e observar cuidadosamente a sessão com o objetivo de obter uma opinião precisa do experimento.

Ao longo da sessão, o tempo para cada exercício e cada fase (resolução e avaliação automática com o Mooshak e visualização / animação da solução correta com o Jeliot) foi estritamente controlado, a fim de garantir que todas as subtarefas pudessem ser executadas durante a aula (2 horas). A primeira meia hora foi usada para preparar os estudantes para a sessão e o fluxograma foi divulgado e explicado. O tempo restante, os 90 minutos, foi dividido em três partes iguais, alocando meia hora para cada exercício, ou seja, 15 minutos para desenvolver e testar uma solução e mais 15 minutos para animar uma solução correta. No final da aula cada aluno preencheu um pequeno inquérito para recolha da opinião individual sobre a experiência.

Em relação ao primeiro objetivo, os professores presentes na sala observaram e afirmaram que ambas as sessões foram executadas com êxito; nenhum incidente foi registado. Todas as tarefas planeadas foram devidamente realizadas.

Em relação ao segundo objetivo, mais uma vez, os observadores relataram que todos os estudantes estiveram completamente envolvidos em terminar as atividades propostas.

Na próxima secção é discutido o terceiro objetivo.

7.1.3 Discussão dos resultados

À primeira vista, e de acordo com os dados recolhidos a partir de Mooshak e resumidos na Tabela 1, pensamos que o comportamento dos estudantes realmente mudou ao longo da aula, e a sua produtividade aumentou, pois foram resolvendo mais facilmente os exercícios propostos. Como pode ser observado na Tabela 1, o número de respostas corretas foi aumentando e, a primeira linha regista o número de submissões para cada exercício que foram completamente aceites pelo Mooshak, isto é, que produziram o resultado esperado para todos os valores dos dados de entrada fornecidos. As linhas dois e três registam o número total de submissões que foram avaliadas pelo Mooshak como erradas (porque: a saída produzida não é o esperado; ou foi detetado um erro de compilação; ou um erro de execução (tempo, ou memória consumida excedida)).

	Ex. 1	Ex. 2	Ex. 3
Nº of correct answers	4	6	9
Nº of Wrong answers	12	11	14
Nº of compilations Errors	30	37	20
Total nº of submissions	46	54	43
Average of submissions	1,6	1,9	1,5
Correct answers after error	1	2	4
Correct answers at 1st sub.	1	4	4

Tabela 7.1: Resumo dos resultados do experimento

É importante observar que um estudante pode submeter o exercício mais do que uma vez até obter uma solução correta. Assim, o total de submissões apresentado na quarta linha mostra toda a atividade dos alunos, a sua persistência, e o grau de dificuldade dos exercícios. As três últimas linhas mostram os detalhes sobre as submissões, a fim de refinar as conclusões que podem ser tiradas a partir de quarta linha.

Assim, uma primeira conclusão é que os estudantes resolveram o terceiro exercício mais rápido (com menor número de submissões) e com melhores resultados (número total de submissões aceites). É importante ressaltar que o último exercício não foi mais fácil do que os anteriores. Assim sendo, na nossa opinião estas conclusões corroboram a nossa hipótese.

Também é importante salientar que os alunos mostraram uma maior dificuldade ao resolver o segundo exercício de acordo com o número de submissões; pensa-se que foi devido ao enunciado do problema que era um pouco mais elaborado. Além da informação numérica

apresentada na Tabela 1, os dois professores presentes na sala de aula também observaram esta evidência. Esta conclusão também suporta a hipótese de que os alunos, por vezes, apresentam grandes dificuldades para entender os enunciados dos problemas. Este comentário dos professores é corroborado pela resposta dos estudantes ao questionário como mostrado na próxima subsecção.

Uma conclusão também retirada foi que o tempo de resolução foi um pouco restritivo. Se os alunos tivessem tido mais tempo para cada um dos exercícios, acreditamos que os resultados seriam, por um lado, mais bem-sucedidos, e por outro lado o processo teria sido mais eficaz e motivador em termos de atividade de aprendizagem. Ou seja, com a nossa observação podemos dizer que a fase de animação beneficiaria se fosse possível atribuir-lhe mais tempo.

Como referido, a evolução do comportamento dos alunos ao longo da aula de duas horas mostrou que esta abordagem levou a um melhor desempenho por parte dos alunos. Por um lado, nota-se que o número de alunos com submissões aceites aumentou e por outro lado, o número de submissões aumentou e o número de erros de compilação diminuiu. Isto significa que a motivação dos estudantes aumentou, enquanto que os erros de base foram reduzindo. A motivação foi uma das nossas principais preocupações. O experimento apresentado também nos permitiu entender a melhor forma de conduzir futuros testes, como por exemplo, uma maior flexibilidade na gestão do tempo durante a aula, como dito acima. Isto significa que temos a intenção de propor os três exercícios no início e permitir que os alunos escolham os intervalos de tempo para usar em cada um deles; desta forma, podem decidir e explorar mais profundamente a animação. Para validar estas conclusões, é importante e necessário repetir o experimento para outros temas, como o processamento de strings ou o processamento de matrizes, envolvendo outras amostras de estudantes.

7.1.4 Opinião dos estudantes

No fim do experimento, cada estudante respondeu a um curto inquérito com três perguntas. A seguir, apresentam-se as questões colocadas e as respostas recorrendo a uma representação gráfica da sua distribuição.

P1. *Teve mais dificuldade na interpretação do enunciado ou na codificação dos exercícios propostos?*

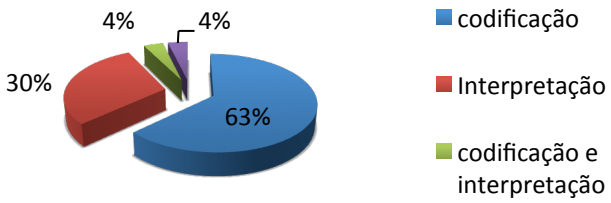


Figura 7.2: Resultados P1

Nesta pergunta, grande parte dos alunos respondeu que a sua maior dificuldade foi na codificação. Porém, 30% dos alunos mostraram dificuldade em entender os enunciados. Como já referido anteriormente a interpretação dos enunciados é um problema efetivo na aprendizagem da programação.

P2. *Até que ponto achou interessante a ferramenta Mooshak?* Saliente dois aspetos.

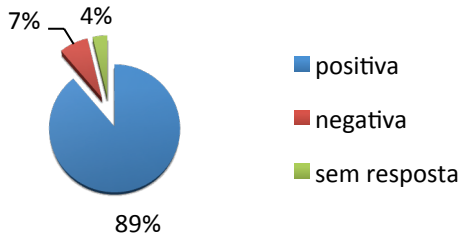


Figura 7.3: Resultados P2

A utilização do Mooshak foi sem dúvida bastante interessante para os alunos. Nas respostas analisadas, a maioria dos alunos responde que foi importante para perceber rapidamente se o exercício está correto ou errado. Os alunos também salientam a importância da apresentação do tipo de erro, tendo a possibilidade de o corrigir e voltar a submeter o exercício.

P3. *Achou importante a animação do algoritmo no Jeliot?* Em que aspetos?

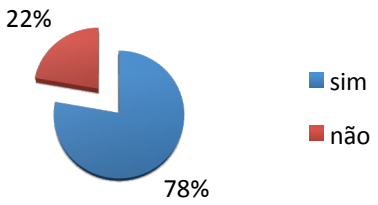


Figura 7.4: Resultados P3

Como se constata, a maioria dos alunos considerou importante a animação oferecida pelo Jeliot. Para esta questão, houve algumas respostas interessantes:

"Sim, pois dá um melhor entendimento de como corre o programa e das suas variáveis."

"Mostra de forma clara e evidente o debug do programa, sendo mais fácil a sua interpretação."

"Sim, para melhor percepção do algoritmo. Após a primeira visualização, percebi muito melhor os exercícios seguintes."

"Sim, cativa mais a atenção."

"Sim, foi importante porque explica bem o problema e resolve-o de uma forma interativa. Ajudou no primeiro exercício a perceber o erro cometido e a resolver corretamente os seguintes exercícios. "

"Sim, porque a resolução interativa/visual facilita a compreensão do exercício."

7.1.5 Sumário

A evolução do comportamento dos alunos ao longo da aula de duas horas mostrou que a abordagem proposta levou-os a um melhor desempenho. Por um lado, observa-se que o número de alunos com submissões aceites aumentou. Por outro lado, o número de aceites aumentou e o número de erros de compilação diminuiu. Isto significa que a motivação dos estudantes aumentou, enquanto os erros de base diminuíram.

O experimento relatado também permitiu entender a melhor forma de realizar testes futuros. Maior flexibilidade na gestão do tempo durante a aula é uma das melhorias que queremos introduzir.

7.2 Concepção e realização de um experimento para aferir a abordagem 2

Neste subcapítulo será apresentada uma descrição do segundo experimento realizado para e aferir a abordagem 2. Neste experimento foi utilizado um grupo de controlo na segunda sessão. Estes alunos resolveram os exercícios propostos sem a ajuda ou apoio de qualquer ferramenta de suporte ao ensino da programação.

7.2.1 Planeamento do experimento.

Para atingir os objectivos acima referidos recorrendo aos recursos de computação disponíveis, é proposto combinar técnicas de animação e avaliação automática. Para concretizar a abordagem e realizar o experimento, será apresentado um conjunto de passos a seguir pelo docente e pelo aluno (reaproveitando o que foi utilizado na primeira sessão):

- Passo 1: Para cada tópico a ensinar, o professor prepara um problema e mais três exercícios semelhantes;
- Passo 2: Inicialmente o aluno utiliza o sistema de animação selecionado, para animar a execução de um programa que resolve corretamente o primeiro desses problemas (pensados no ponto 1), a fim de analisar com cuidado e entender a solução correta e o seu comportamento;
- Passo 3: Para o primeiro exercício, o professor pede ao aluno para o resolver e para testar a solução produzida usando o Sistema de Avaliação de Automático, AA, selecionado. O professor também pode discutir com cada aluno o feedback recebido do sistema AA;
- Passo 4: O professor fornece uma proposta de solução do exercício e o aluno deve utilizar o sistema de animação selecionado, para animar a execução do programa a fim de analisar com cuidado e entender a solução correta e seu comportamento;
- Passo 5: Repetir os passos 3 e 4 para as restantes exercícios.

Esta abordagem pressupõe que o professor selecione uma ferramenta de animação eficaz e fácil de usar, bem como escolha um avaliador automático que além de amigável retorne um

feedback tão detalhado quanto possível. Tal como no 1º experimento, as ferramentas escolhidas para este serão novamente o Jeliot e o Mooshak.

Para este caso específico, para ensinar o tema da matéria de introdução à Programação "arrays", foram desenvolvidos três enunciados, com o cuidado de serem bastante semelhantes:

a) Escreva um programa Java que, dado um número inteiro N, maior que 0, leia um vetor (array unidimensional) de N elementos inteiros e imprima quantos valores pares existem no vetor que sejam maior ou igual ao último valor lido.
b) Escreva um programa Java que, dado um número inteiro N, maior que 0, leia um vetor de N elementos inteiros e transforme o vetor inicial dividindo todos os seus elementos pelo menor valor do vetor. Mostre o vetor após os cálculos, imprimindo um número por linha.
c) Escreva um programa Java que leia um vetor de números inteiros positivos. A introdução de dados termina quando for lido o número 0, sabendo-se que nunca existirão mais de 1000 valores. Em seguida, troque o primeiro elemento com o último, o segundo com o penúltimo, o terceiro com o antepenúltimo, e assim sucessivamente. Imprima o vetor depois da troca.

Depois de decididas as ferramentas concretas a usar, o tema da aula experimental, e os exercícios para resolver, foi necessário escrever um plano cuidado da aula, para que todos os alunos conseguissem entender para o que são convidados a fazer e como devem proceder. Para o efeito, foi esboçado um plano de aula com uma tarefa composta das seguintes etapas:

1) Animação de um programa proposto pelo professor para resolver o primeiro problema com a ferramenta Jeliot;
2) Entrar no sistema Mooshak (AES);
3) Escolha do exercício (A ou B ou C de acordo com a respetiva fase);
4) Ler atentamente o enunciado;
5) Desenvolver um algoritmo e codifica-lo em Java, a fim de resolver o exercício proposto (usar o NetBeans IDE);
6) Submeter o programa escrito, para ser avaliado pelo Mooshak;
7) Analisar a avaliação apresentada pelo Mooshak;

8) Se o resultado não for "Accepted", voltar ao passo 4 e fazer as correções necessárias;
9) Se o resultado for "Accepted", passar para a visualização e animação do programa proposto pelo professor para resolver o problema com a ferramenta Jeliot;
10) Se o exercício atual é o terceiro, C, terminar o experimento; se não, voltar para o passo 2.

Este plano foi desenhado como um fluxograma tendo sido impresso e distribuído a todos os alunos presentes na aula, como o apresentado na Figura 7.2.

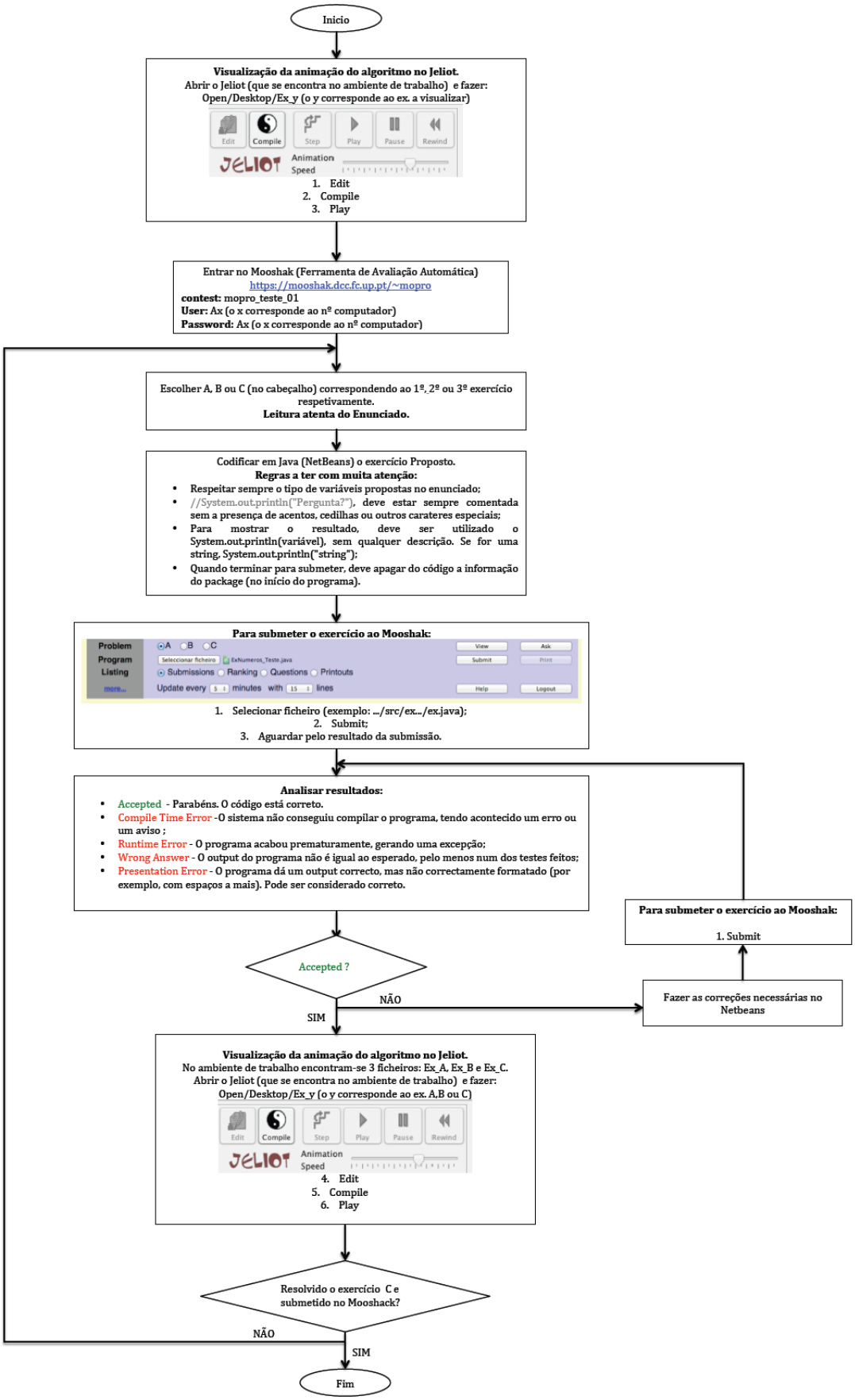


Figura 7.5: Fluxograma representativo do experimento (abordagem 2): visualizar a animação de um algoritmo no Jeliot, resolver três problemas em Java, obter feedback e visualizar a respetiva animação do algoritmo

7.2.2 Condução do experimento

O experimento foi feito com 19 alunos do 1º ano (num curso de Engenharia). Cada sessão foi supervisionada por dois professores que permaneceram todo o tempo na sala de aula para ajudar os alunos, e observar cuidadosamente a sessão com o objetivo de obter uma opinião precisa do experimento.

A primeira sessão foi realizada com 8 alunos, em que o tempo para cada exercício e cada fase (resolução e avaliação automática com o Mooshak e visualização / animação da solução correta com o Jeliot) foi estritamente controlado, a fim de garantir que todas as subtarefas pudessem ser executadas durante a aula (2 horas). Os primeiros quinze minutos foram usados para a animação de um exercício proposto pelo professor. Os quinze minutos seguintes foram utilizados para preparar os estudantes para a sessão e o fluxograma foi divulgado e explicado. O tempo restante, os 90 minutos, foi dividido em três partes iguais, alocando meia hora para cada exercício, ou seja, 15 minutos para desenvolver e testar uma solução e mais 15 minutos para animar uma solução correta. No final da aula cada aluno preencheu um pequeno inquérito para recolha da opinião individual sobre a experiência.

A segunda sessão foi realizada com 11 alunos, em que o tempo para cada exercício foi controlado a fim de garantir que todos os exercícios fossem resolvidos dentro do tempo estipulado (sem recorrer a qualquer ferramenta de Animação e Avaliação Automática).

Ambos os experimentos foram executados com êxito e nenhum incidente foi registado, sendo todas as tarefas planeadas devidamente realizadas. Em relação ao segundo objetivo, observar se os alunos estão envolvidos e motivados, mais uma vez os observadores relataram que todos os estudantes estiveram completamente envolvidos em terminar as atividades propostas.

7.2.3 Discussão dos resultados

Na primeira sessão e de acordo com os dados recolhidos a partir de Mooshak pensamos que o comportamento dos alunos melhorou ligeiramente ao longo da aula, sendo que no primeiro exercício houve 100% de erradas e no segundo 75%, não se verificando desistências na tentativa de resolução. No terceiro exercício não se verificou qualquer melhoria, que na nossa opinião, se deveu a um maior grau de dificuldade desse mesmo exercício.

Para a segunda sessão (grupo de controlo) verificou-se que não houve qualquer evolução, ou seja, quem acertou na solução foram sempre os mesmos alunos, com 30% dos alunos com respostas corretas e 70% com respostas erradas. Também se constatou que, perante o código apresentado pelo aluno, as tentativas de resolução foram diminuindo, e que no último exercício apresentado apenas 20% dos alunos tentaram resolver o exercício. Como no primeiro exercício 60% dos alunos tentaram uma resolução, constata-se que houve um rápido desinteresse dos alunos.

Também é importante salientar que os alunos mostraram uma maior dificuldade ao resolver exercícios com arrays, tendo essas dificuldades incidido sobretudo na tarefa de codificação. Além da informação numérica apresentada, os dois professores presentes na sala de aula também observaram estas evidências. Este comentário dos professores é corroborado pela resposta dos estudantes ao questionário como mostrado na próxima subsecção.

7.2.4 Opinião dos estudantes

Para a primeira sessão, no fim do experimento, cada estudante respondeu a um curto inquérito com três perguntas. Abaixo, apresentamos as questões colocadas e os resultados recolhidos, tendo-se escolhido uma representação gráfica da distribuição de respostas.

P1. *Teve mais dificuldade na interpretação do enunciado ou na codificação dos exercícios propostos?*

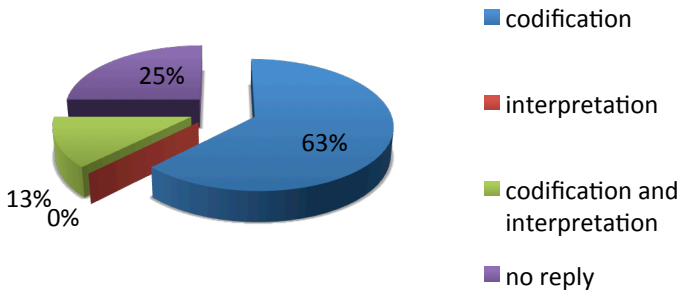


Figura 7.6: Resultados P1

Nesta pergunta, grande parte dos alunos (63%) respondeu que a sua maior dificuldade foi na codificação.

P2. *Até que ponto achou interessante a ferramenta Mooshak?* Saliente dois aspetos.

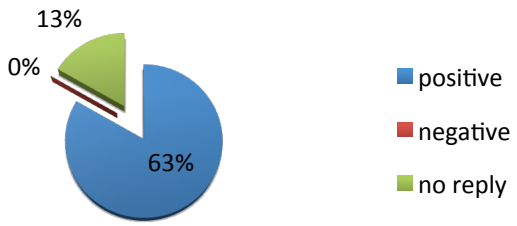


Figura 7.7: Resultados P2

Como se observa na Figura 7.7, 63% do alunos respondeu de novo que a utilização do Mooshak foi bastante interessante.

P3. *Achou importante a animação do algoritmo no Jeliot?* Em que aspetos?

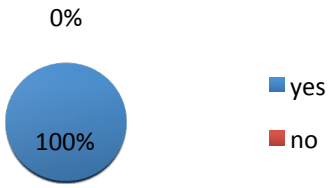


Figura 7.8: Resultados P3

Como se constata, 100% dos alunos consideraram importante a animação oferecida pelo Jeliot.

7.2.5 Sumário

A evolução do comportamento dos alunos na primeira sessão, do experimento que seguiu a abordagem 2 suportada por ferramentas, ao longo da aula de duas horas mostrou que a abordagem proposta levou-os a um ligeiro aumento de envolvimento e desempenho. Por outro lado, o grupo de controlo, que teve uma sessão tradicional sem qualquer suporte extra, não mostrou qualquer entusiasmo, piorando o seu contributo ao longo da aula.

O experimento relatado também permitiu entender a melhor forma de realizar testes futuros. Maior flexibilidade na gestão do tempo durante a aula é uma das melhorias que queremos introduzir. Além disso, ficou-se com a ideia de que a sessão inicial de aprendizagem baseada na análise da animação de programas corretos devia ser mais longa e com mais exemplos.

8. PEP: PLATAFORMA PARA SUPORTE AO ENSINO DA PROGRAMAÇÃO

Como referido nos capítulos anteriores, a motivação principal deste trabalho de doutoramento reside na dificuldade do processo ensino/aprendizagem da programação e seu insucesso. A plataforma que se apresenta neste capítulo tem como objetivo principal diminuir as dificuldades sentidas pelos alunos que se encontram numa fase inicial do ciclo de aprendizagem de programação. Estas dificuldades incentivam desde há alguns anos a elaboração de novas ferramentas que permitam facilitar todo o processo de transmissão de conhecimento sobre os elementos básicos da programação, envolvendo os alunos e capacitando-os para a resolução de problemas por computador.

Recorde-se que, com vista a minorar as dificuldades que os alunos de informática sentem nas disciplinas de introdução à programação e face aos recursos informáticos atualmente disponíveis, propuseram-se no Capítulo 6 duas abordagens que fazem uso de técnicas de animação de programas e de sistemas de avaliação automática de programas com vista a aumentar o envolvimento dos alunos a sua auto-confiança e por conseguinte aumentar a

motivação essencial à aprendizagem eficaz destas matérias. Nesse contexto será desejável ter uma plataforma baseada na Web que apoie a aplicação real dessas abordagens.

Seguindo a abordagem 2, apresentada no Capítulo 6, foi desenvolvida uma ferramenta baseada na Web (designado por PEP) que permitirá apoiar o professor nas aulas laboratoriais e sobretudo facultará aos alunos a possibilidade de fazerem sessões de estudo fora das aulas. Tal sistema irá permitir: (i) *ao docente*, carregar e manter os exercícios (organizados por temas e dificuldades) a usar em cada sessão, bem como fazer o plano das sessões; (ii) *ao aluno*, executar uma ou mais sessões, para praticar um determinado tema do curso, animando os exercícios e depois resolvendo-os e testando-os com recolha imediata de *feedback*. A plataforma PEP vai ainda permitir que o professor consiga visualizar a informação sobre a forma como decorreu cada sessão de trabalho de cada aluno (data e hora, sequência de exercícios resolvidos, tempo gasto, etc.).

Seguindo os requisitos expostos e a mencionada abordagem 2, um grupo de alunos do Mestrado de Engenharia Informática da UM (MEI), no âmbito do seu projeto integrado do 1º ano, desenvolveu o sistema pretendido e que se apresenta nas secções seguintes.

8.1 Arquitetura

Conforme se pode ver na Figura 8.1, o sistema é formado por duas grandes componentes: o *Back-office* (BO) e o *Front-office* (FO).

O primeiro (*Back-office*) suporta todas as tarefas de gestão da base de dados que armazena os exercícios e as *queries* que serão posteriormente usadas pelo segundo módulo para construir as sessões que serão apresentadas aos alunos. No BO apenas os professores terão acesso e é a partir daí que podem gerir exercícios (criar, editar e apagar), planear sessões, bem como analisar os dados recolhidos pelo segundo módulo relativos às sessões realizadas pelos alunos.

Além disso, o BO tem mais duas componentes essenciais: o **compilador** que lê a especificação formal de cada sessão (escrita numa linguagem de domínio específico, DSL, que foi criada especificamente para este propósito) e gera o código necessário para o FO montar as sessões; e o **módulo de análise** que recupera da base de dados a informação relativa a cada sessão de

cada aluno e a apresenta ao professor para que este possa acompanhar o processo de aprendizagem.

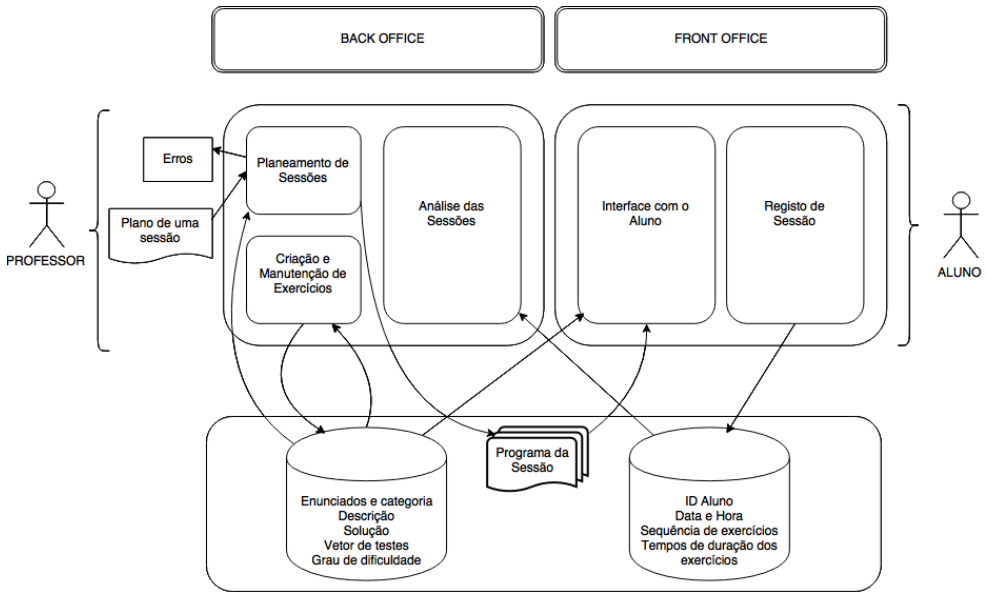


Figura 8.1: Arquitetura do Sistema PEP

Quanto ao *Front-office*, é destinado aos alunos. Nesse módulo as sessões serão apresentadas aos alunos de modo a que eles possam escolher uma delas dentro de um tema que queiram (ou tenham de) exercitar e depois possam realizar o dito estudo. O FO também se encarrega de gravar as informações sobre cada sessão, ou seja, a identificação do aluno que realiza a sessão, a data, a sequência de exercícios resolvidos, bem como o tempo gasto a trabalhar em cada um deles. Todas estas informações serão guardadas na base de dados para mais tarde o professor as poder analisar através do módulo apropriado do BO como referido atrás.

8.1.1 Tecnologias usadas na implementação

Para concretizar o PEP, foram utilizadas tecnologias essenciais para o seu desenvolvimento como por exemplo:

- *yiiFramework 2.0*, um *framework* de alta performance em PHP que utiliza componentes para o desenvolvimento de pequenas/médias/grandes aplicações Web. Esta *framework* permite a reutilização de código na programação Web e pode acelerar significativamente o processo de desenvolvimento. O nome Yii representa as palavras Yes It Is! (Sim isto é!).

- *ANTLR* (*ANother Tool for Language Recognition*), é um poderoso sistema para geração automática de compiladores (ou melhor, de processadores de linguagens) para leitura de frases de uma linguagem formal (neste caso, a linguagem especificamente criada para descrever as sessões de estudo), seu reconhecimento (análise) e posterior tradução. A partir de uma gramática da linguagem pretendida, ANTLR gera um analisador sintático que pode construir e percorrer a árvore de derivação de modo a fazer a análise semântica da dita frase e sua transformação para produzir a saída desejada.
- *Jeliot,* é o sistema de visualização/animação de programas em Java apresentado no Capítulo 3.
- *Mooshak* é o sistema para gerir concursos de programação na Web usado para correção automática de programas descrito no Capítulo 5.

Combinando estas ferramentas foi possível então desenvolver a plataforma de suporte ao ensino da programação PEP.

8.2 PEP na perspectiva do Aluno

Na Figura 8.2, está representado um esquema que mostra o grafo que define o fluxo de trabalho (a sequência de tarefas) usando a Interface apresentada pelo PEP ao aluno. Numa primeira etapa está implementado um mecanismo simples para controlo de acessos. Para o usar é necessário o prévio registo na plataforma por parte do aluno e a partir daí este já pode fazer a sua autenticação com a finalidade de entrar e obter acesso às sessões de estudo (Figura 8.3). Após a escolha da sessão a realizar (Figura 8.4), o utilizador será confrontado com duas novas opções de escolha, como visível na Figura 8.5: seguir para a 1ª parte que usa técnicas de animação de programas; ou seguir para a 2ª parte da avaliação automática de programas (aconselhada só após realizar as tarefas da parte 1, ou então se já for aluno mais experiente e já se sinta capaz de ir diretamente resolver exercícios). Caso escolha a Parte 1 (Animação), o utilizador tem acesso a uma descrição do problema bem como a solução do mesmo (Figura 8.6). Poderá então usar a ferramenta Jeliot para animar a resolução do problema em causa para melhor compreensão. Na parte 2 (avaliação), está presente apenas uma descrição de um problema e incentiva-se a sua resolução (Figura 8.7). Aqui a ferramenta

de Avaliação Automática escolhida (Mooshak) é usada para que os alunos possam verificar se a sua resolução está correta ou não.

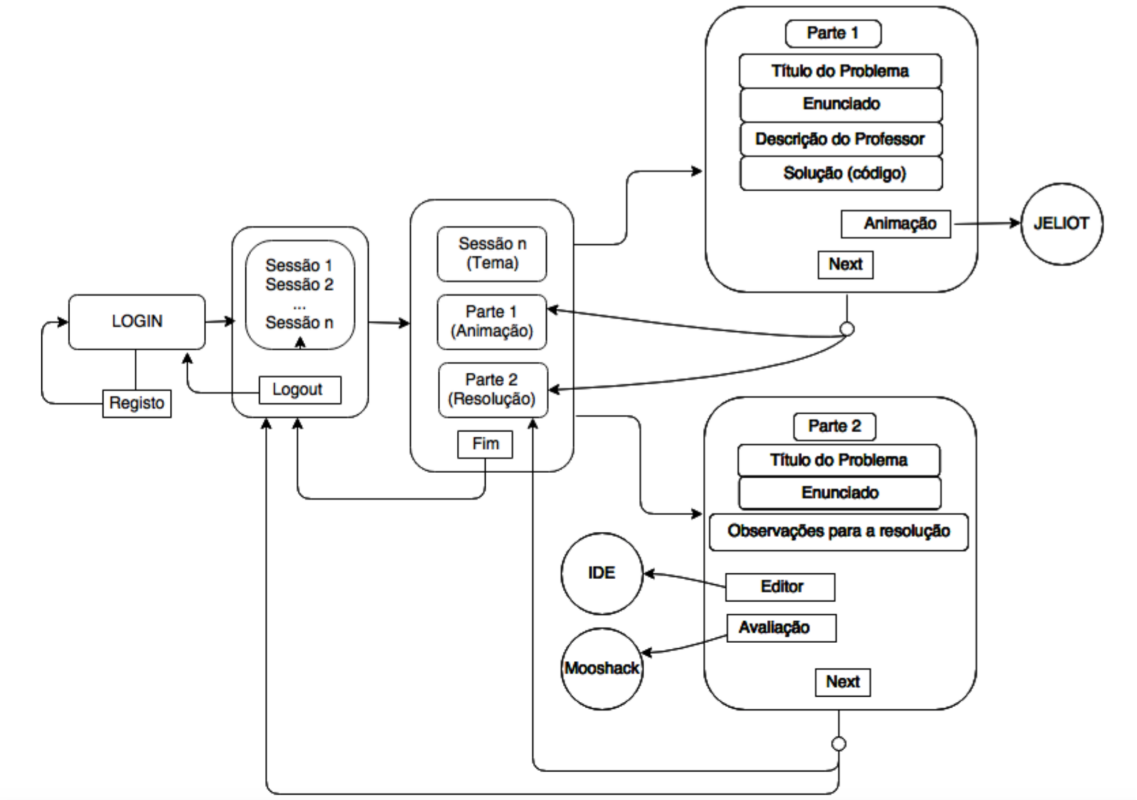


Figura 8.2: Planificação do Sistema

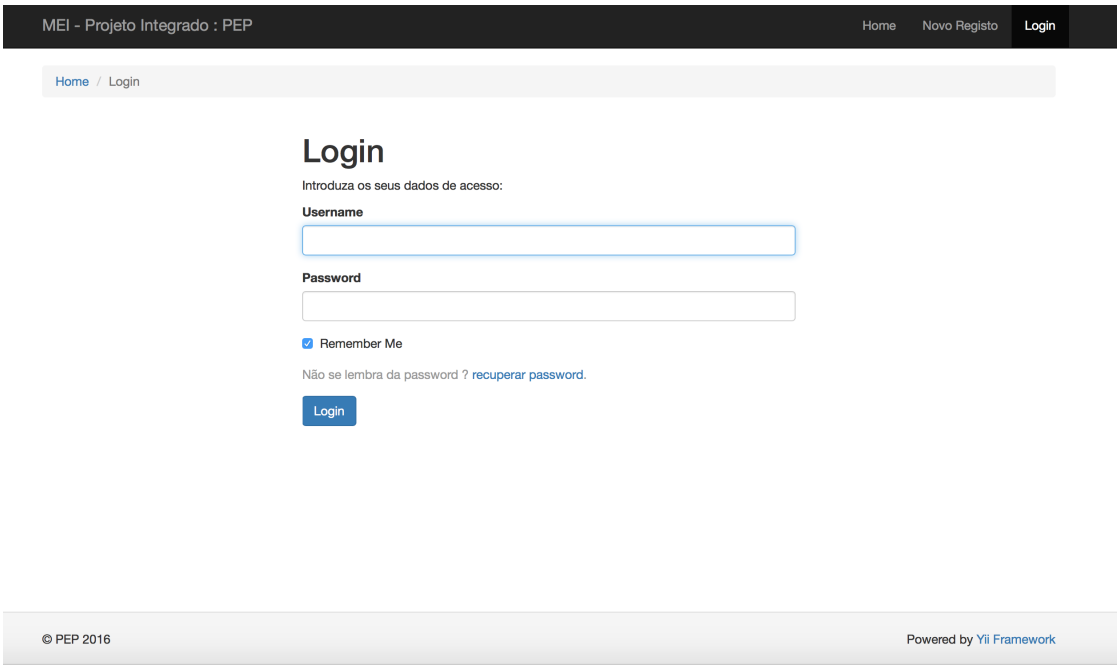


Figura 8.3: Página de Login do aluno

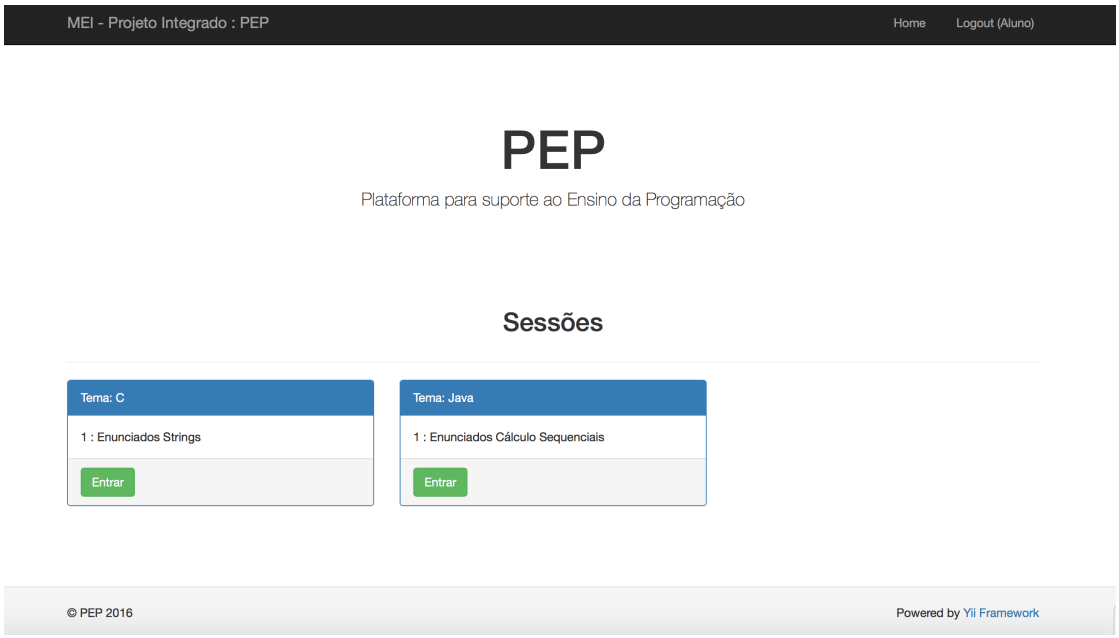


Figura 8.4: Página da listagem das sessões

Figura 8.5: Página para a escolha da parte1 ou parte2

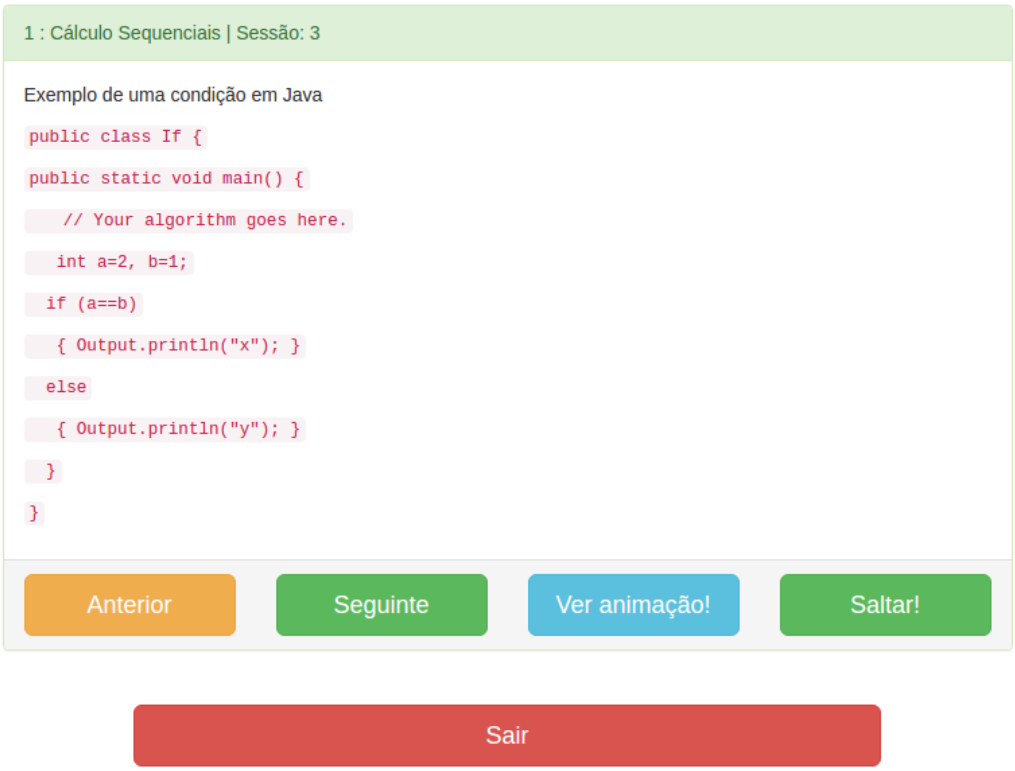


Figura 8.6: Página para visualização da solução e possibilidade da Animação

Figura 8.7: Página com a questão e possibilidade de Avaliação Automática

8.3 PEP na Perspetiva do Professor

Apenas os Professores terão acesso à componente Back-Office, podendo gerir os exercícios (criar, editar e apagar), planear sessões, bem como analisar as sessões submetidas pelos alunos.

8.3.1 Base de Dados

Na Figura 8.8 encontra-se representada a estrutura da Base de Dados desenvolvida para o PEP, desenhada para ser simples e eficaz. A sua estrutura é composta por quatro tabelas: User, Atividade, Questão e Tema.

A tabela User é criada automaticamente através da *yiiFramework*. Esta guarda todos os utilizadores que se registaram no sistema.

A tabela Tema é destinada a guardar os temas das sessões, cada tema acompanhado de uma breve descrição. Cada sessão tem um tema, e é nesta tabela que é feita a verificação de existência de tal tema. Contém ainda as linguagens de programação que o docente pretende ensinar. Por exemplo, ao Tema Java estariam associadas várias sessões em que os nomes poderiam ser: "Strings", "Ciclos", etc.

Na tabela Questao existem os campos necessários para guardar informação relativamente a uma pergunta/enunciado/demonstração/exercício.

- tipo - Se a questão é da parte 1 ou 2. (utilizada validação das sessões);
- dificuldade - Grau de dificuldade que o professor associa à pergunta (utilizada para verificar se numa parte de uma sessão, as questões estão por ordem de dificuldade);
- enunciado - Contém o título do problema, o enunciado e possibilita também uma breve descrição ou observações para a sua resolução;
- solução - Contém a solução do problema;
- tempo - tempo que o professor considera aceitável para a resolução do problema;
- tema - É associado a cada questão um tema presente na tabela Tema através do seu id;
- multimedia - Caso o professor queira recorrer a uma imagem/vídeo para ajudar na descrição da questão;

- ajuda - Ajuda em forma de dica para a resolução da questão (dada pelo professor e destinada apenas a questões do tipo 2).

Por último, a tabela Actividade contém o campo *utilizador* que está relacionado com o campo *id* da tabela User, o campo questão que está relacionado com o campo *idQuestao* da tabela Questao. Estes relacionamentos são necessários para saber a quem pertence tal atividade bem como a qual questão. Existente ainda os campos *resposta* para guardar as respostas inseridas pelos alunos, o campo *avaliação* para registar o resultado obtido através do Mooshak, o campo *data* e tempo para mais tarde ser possível analisar em que dia foi efetuada a atividade bem quanto tempo demorou. Já o campo *opção* está destinado para guardar cada decisão que o aluno tomou durante cada questão, ou seja, se seguiu em frente, se voltou atrás, se fez a animação com o Jeliot, se submeteu no Mooshak e se recorreu à ajuda ou à resolução. O campo *sessão* serve para identificar que sessão o aluno está a consultar.

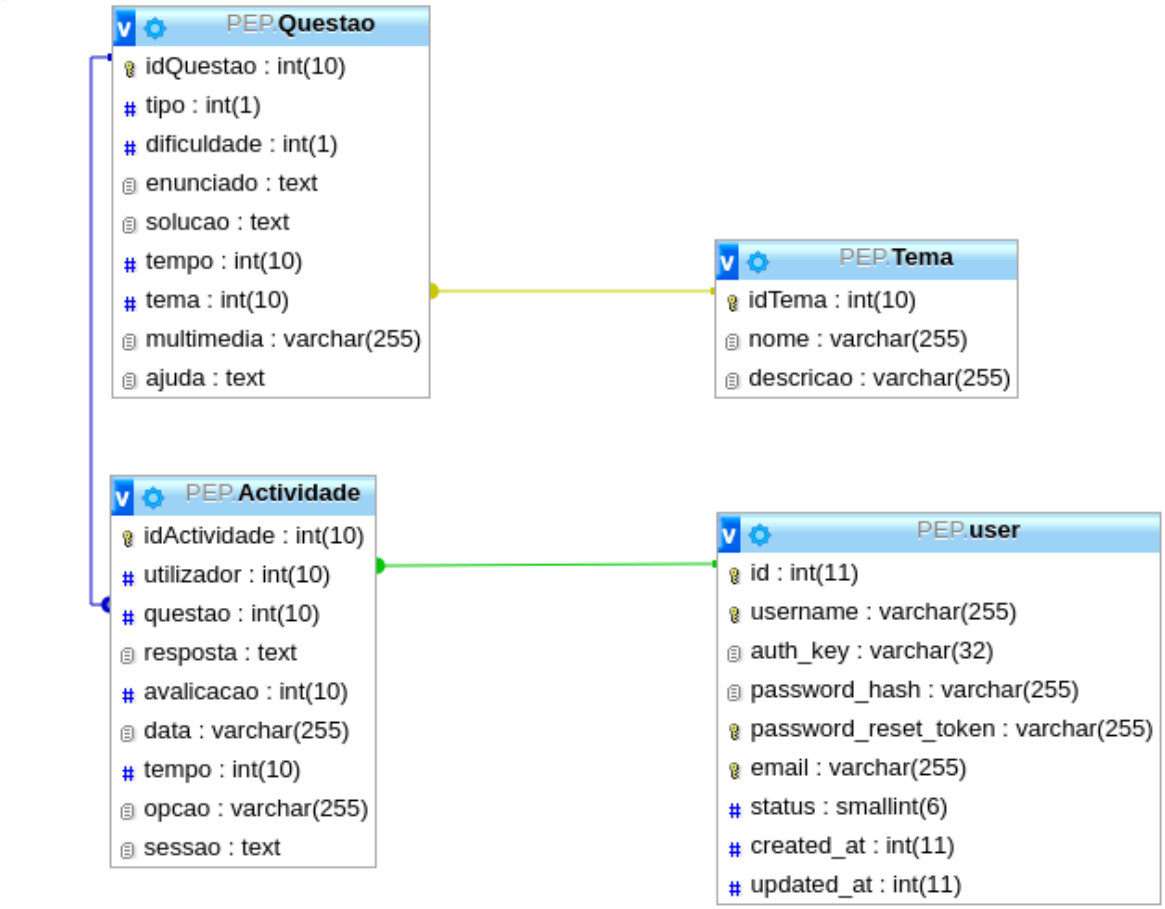


Figura 8.8: Estrutura da Base de Dados

8.3.2 Compilador

A principal funcionalidade do compilador desenvolvido é validar as especificações das sessões definidas pelos docentes (como mostra a Figura 8.9), ou seja, se estão com dados válidos e existentes na base de dados.

```
SESSOES
    "Java" [
        s1 : "Enunciados Cálculo Sequenciais" ,
        Parte1 : p1 (15) NOJUMP ; p2 (10) JUMP ; p3 (10) JUMP ; p4 (10) NOJUMP,
        Parte2 : p1 (15) NOJUMP ; p2 (10) JUMP ; p3 (10) JUMP ; p4 (10) NOJUMP
    ]
    "C" [
        s1 : "Enunciados Strings" ,
        Parte1 : p1 (15) NOJUMP ; p2 (10) JUMP ; p3 (10) JUMP ; p4 (10) NOJUMP,
        Parte2 : p1 (15) NOJUMP ; p2 (10) JUMP ; p3 (10) JUMP ; p4 (10) NOJUMP
    ]
FIM
```

Figura 8.9: Estrutura do ficheiro de entrada

O compilador, começa por ler o texto (escrito na linguagem específica definida pela gramática Figura 8.10) e após reconhecer a frase, faz uma ligação à base de dados para recolher informação relativa a temas e questões nela inseridas e verifica se o tema da sessão está inserido na Base de Dados, se o número identificativo das questões também se encontra na base de dados, se o grau de dificuldade dos problemas está por ordem crescente, se o docente indicou o tempo para cada exercício, bem como se pode ou não avançar para a parte seguinte. Caso não sejam verificadas estas condições, o docente é informado dos erros sendo-lhe indicada de que questão se trata e que problemas ocorreram, de modo a poder rever a sua especificação. Por fim, se tudo correr bem, sem erros, será criado um ficheiro com instruções para montar a mesma sessão do ponto de vista do Aluno no *Front-office*.

```
plano : INICIO listaTemas ']' FIM
      ;
listaTemas : tema ( ']' tema )*
           ;
tema : PAL '[' listaSessoes
     ;
listaSessoes : sessoes ('.' sessoes)*
             ;
sessoes : sessao ',' parte1 ',' parte2
        ;
sessao : id ':' PAL
       ;
parte1 :  id ':' (pergunta tempo salto ';'* )+
       ;
parte2 : id ':' (pergunta tempo salto ';'*) +
       ;
pergunta : id
         ;
tempo : '('NUM')'
      |
    ;
salto : 'JUMP'
      | 'NOJUMP'
      |
      ;
id : 's' NUM
   | 'p' NUM
   | 'Parte' NUM
   | 't' NUM
   ;
INICIO : 'SESSOES';
SEP : ',';
FIM : 'FIM';
PAL : '"'[a-zA-Z0-9\.-\,()?=:\\|í|é|º|ç|ã|á|à|ó|ú|â|ª| /]*'"' ;
NUM : [0-9]+;
WS : ('\r'?'\n'|' '|'\t')+ -> skip;
```

Figura 8.10: Gramática elaborada

8.3.3 Manutenção da Base de Dados

Como já foi mencionado, o BO do PEP possibilita a gestão dos temas através das tabelas Tema e Questão (Figura 8.11), bem como a criação de sessões através de um campo de texto ou submissão de ficheiros. Também é possível ao Docente consultar as atividades realizadas pelos alunos. Na Figura 8.12 é possível observar a página de entrada na aplicação de gestão da BD; nela se podem ver os 4 botões Tema, Questão, Sessões e Resultados que dão acesso a uma página que mostra os registos já existentes na respetiva tabela da base dados dado então possibilidade de acrescentar novos registos, editar os atuais ou mesmo eliminá-los. Caso o clique seja efetuado no botão "Criar sessões", é reencaminhado para uma página (Figura 8.13) onde existe a possibilidade de submeter um texto/ficheiro com a especificação de uma sessão para ser submetida ao Compilador, conforme detalhado na secção anterior. Por fim, o botão "Ver resultados" mostra uma página com o registo de todas as atividades existentes até ao momento.

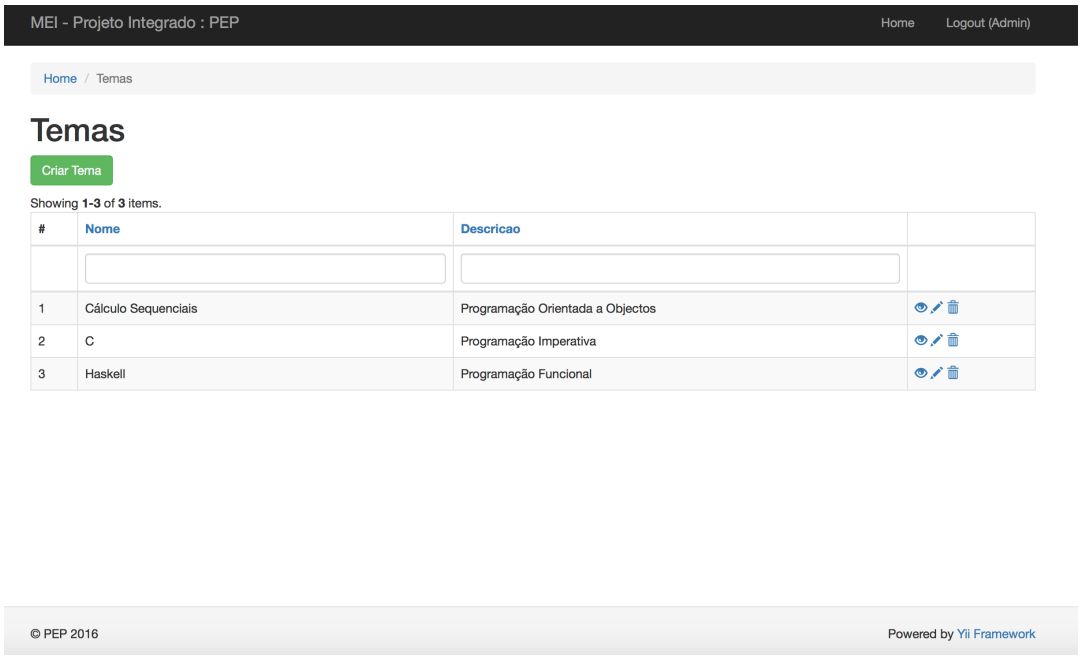


Figura 8.11: Página para gestão dos temas

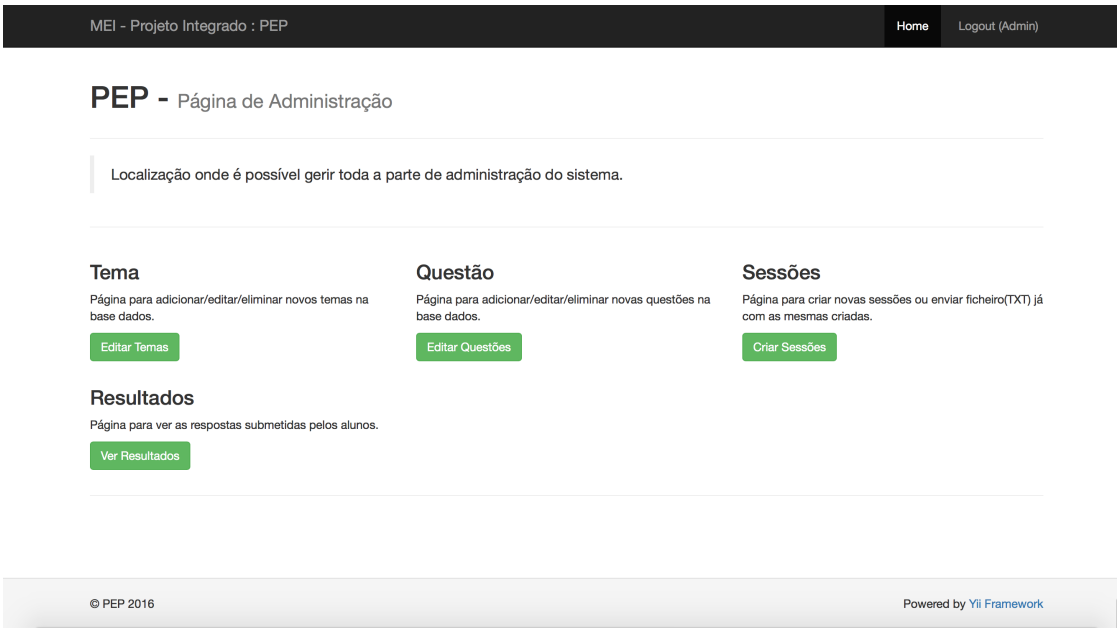


Figura 8.12: Home Page do Back Office

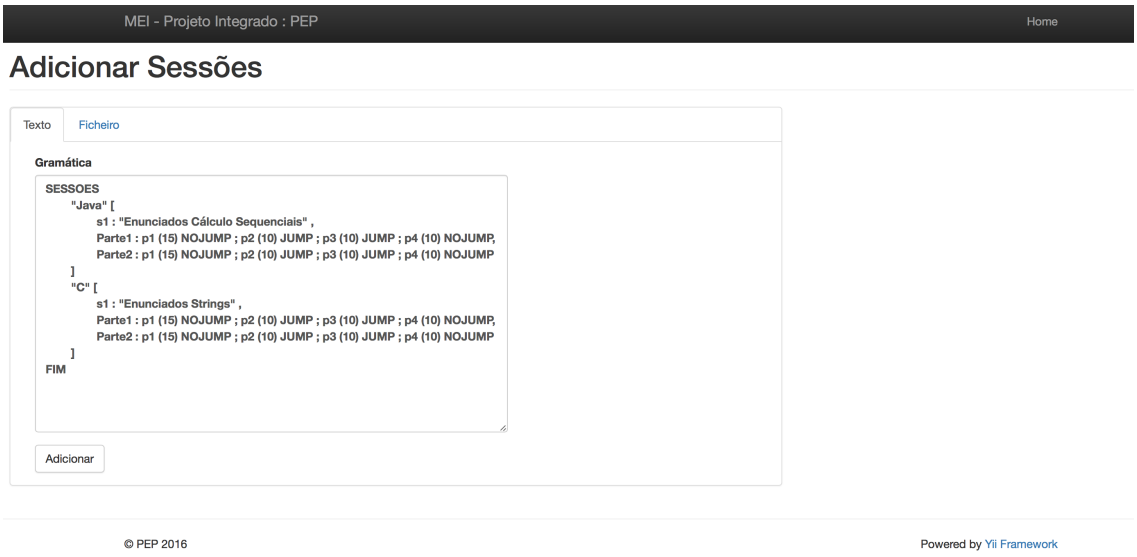


Figura 8.13: Página para gestão das sessões

8.4 Sumário

PEP é uma ferramenta fácil de usar, acessível como um aplicativo da Web, que pode ser usado para aumentar a motivação das atividades de auto-estudo.

Apesar de se acreditar firmemente que o PEP irá melhorar o processo de aprendizagem global, pretende-se melhorar a plataforma introduzindo-lhe alguns dos artefactos de gamificação apresentados na Secção 2.4.1 com vista a aumentar a motivação dos alunos. Nomeadamente acredita-se que a compensação aos alunos como a atribuição de pontos e medalhas devem ser incluídos para premiar os exercícios tentados e os bem resolvidos dentro ou fora da aula (poderão ter pesos diferentes), sendo de incluir o tempo gasto nessa pontuação. No mínimo, uma barra de progresso deve ser adicionada à interface do sistema, estando-se a ponderar se é de incluir um ranking que mostre a posição do aluno face ao resto da turma. Motivados por receber maior pontuação e reconhecer um mais rápido progresso, os alunos serão levados a praticar mais vezes e a desenvolver melhores soluções. Níveis e missões já fazem parte, desde a origem, do desenho do PEP; o recurso a narrativas que contextualizem cada problema de novo a envolver mais o aluno, é já prática corrente e será sempre tido em consideração na escrita dos enunciados. Por fim, a possibilidade (explícita e controlada) de partilha de soluções entre os alunos e a sua discussão em fórum apropriado, deverá vir a ser contemplada.

9. AVALIAÇÃO DO PEP

O uso de *software* educacional tornou-se muito comum. Atualmente existe uma grande variedade de exercícios e práticas, jogos para o ensino, sistemas de aprendizagem integrados, sistemas de resolução de problemas, sistemas de simulação, tutoriais, etc, a maioria deles baseado na Web (Espinoza et al., 2006). Esta diversidade aumenta a preocupação na melhoria da qualidade dos sistemas suportados no computador para apoio do ensino levando a níveis de exigência elevados. Neste contexto, torna-se cada vez mais necessário desenvolver métodos de análise e avaliação da qualidade destes materiais (Escudeiro, 2007).

Nesse sentido, a plataforma de apoio ao ensino da programação (PEP), que se concebeu como resultado de uma das abordagens pedagócicas proposta no âmbito desta tese de doutoramento, não poderá nem deverá escapar a uma tal avaliação que permita aferir a sua qualidade didática.

Para tal fez-se um estudo das abordagens mais consensuais à avaliação da qualidade de software educativo, que se sumaria na secção 9.1, e depois aprofundou-se o estudo da *framework Quantitative Evaluation Framework* (QEF) desenvolvida e em uso no ISEP, conforme também apresentado resumidamente em 9.2, a qual foi escolhida para avaliar o protótipo e que é sugerida para guiar o desenvolvimento de uma versão final que melhore a atual.

9.1 Avaliação do *Software* Educativo

A avaliação do *software* educativo é muito importante para aferir se o *software* é utilizável e se está de acordo com aquilo que os seus utilizadores esperam. Para esta avaliação é necessário um processo sistemático de recolha de dados responsáveis por informar como é que os utilizadores devem utilizar o produto, para uma determinada tarefa num determinado ambiente. A avaliação de um sistema de *software* tem três objetivos, tais como (Webber et al., 2009):

1. Avaliar a funcionalidade desse sistema (o que permite fazer, o que oferece);
2. Avaliar o efeito ou impacto que o seu uso tem sobre os alunos;
3. Identificar falhas na execução aos diversos níveis.

Esta avaliação deve ser feita tanto na fase de desenvolvimento como na fase da utilização do *software*.

A avaliação deve garantir que o *software* satisfaz os requisitos dos alunos, é fácil de aprender, eficaz, seguro, desafiador, com uma linguagem adequada e que estimula a criatividade. Para esta avaliação ser completa e fiável, é importante considerar certas características (ou requisitos) que se enumeram a seguir (Webber et al., 2009):

- Pedagógica – conjunto de características que medem o valor didático da utilização do *software*, como por exemplo ambiente educacional, importância relativamente ao programa curricular, e os aspetos didáticos;
- Usabilidade – Atributos que determinam a facilidade e a objetividade de uso do *software*;
- Interface – facilidade na interação do aluno com o *software*;
- Adaptabilidade – capacidade do *software* de se adaptar ao utilizador e de se adaptarem a diferentes situações e temas;
- Funcionalidade – conjunto das tarefas que se podem realizar;
- Documentação – Característica relativa ao material que acompanha o *software* que permite avaliar a sua completude, legibilidade, organização, etc., de maneira que os alunos entendam de forma clara o que lhe é proposto;

- Portabilidade – determina a disponibilidade do *software* para ser usado em sistemas operativos diferentes.

Segundo Rosa Reis (Reis, 2016), a avaliação da qualidade de um *software* deve ser analisada em função de diferentes pontos de vista. Há necessidade de formar uma equipa multidisciplinar, para se observar os diferentes aspetos acima referidos. Essa equipa será constituída: pelo Professor, para avaliação dos aspetos didáticos, pelos Alunos (como utilizadores) que avaliam a usabilidade e a funcionalidade e por último pela Equipa de desenvolvimento, que avalia a adaptabilidade, interoperabilidade e segurança de acesso, de modo que cada um deles possa exprimir o seu ponto de vista relativamente aos requisitos acima enumerados.

Segundo N. Espinoza (Espinoza et al., 2006), uma metodologia bem definida e desenvolvida sobre Avaliação de *software* Educacional pode ajudar os professores na hora de escolher o *software* a usar. O seu uso deve ser acompanhado de uma revisão crítica do mesmo pois, muitas vezes, estas ferramentas fornecem conclusões pouco claras. A metodologia de avaliação deve ser simples e concreta para ser utilizada por todos os professores adequando-se a todos os níveis e a todas as matérias. Devem existir diferentes técnicas e vários critérios de avaliação. Por exemplo:

a) Na perspetiva de como se faz, temos:
    - Observação Direta
        - Técnica
        - Didática
    - Experimentação
b) Na perspetiva de quem a faz:
    - Individual
    - Colaborativa
c) Na perspetiva de como se recolhe os dados:
    - Por questionário
    - Por observação
        - Objetiva
        - Subjetiva

Estas diferentes técnicas permitem ao professor escolher o *software* para as suas aulas segundo as suas preferências e objetivos. A escolha de cada técnica depende do critério considerado. Características que podem facilitar a aprendizagem e a usabilidade podem ser importantes para uns e não para outros. É importante que a escolha do *software* (metodologias de avaliação diferentes) considere a área de aplicação do *software* a ser avaliado. Os requisitos definidos para a educação não podem ser os mesmos para um produto, por exemplo, direcionado para a medicina. Os professores devem avaliar o *software* escolhido, pois a sua escolha está diretamente relacionada com as características dos seus utilizadores (poderem conjugar os elementos e critérios de acordo com suas necessidades) (Reis, 2016; Espinoza et al., 2006).

Vários autores (Escudeiro, 2007; Godoi et al., 2009; Silva, 2002) propõem um conjunto de critérios de diferente natureza baseados em aspetos (ou dimensões) ergonómicos, pedagógicos e técnicos, ou outros que sejam importantes e pertinentes (ex.: comunicacionais). A *dimensão do domínio ergonómico* garante que o *software* é usado com segurança, conforto e produtividade. A *dimensão do domínio técnico*, diz respeito às características do conteúdo digital relativamente aos aspetos operacionais. Por último, a *dimensão do domínio pedagógico* tem como referência a aprendizagem, garantindo que a relação aluno/instrumento pedagógico é assegurada (estratégias didáticas e tarefas cognitivas estejam em concordância).

Cada dimensão agrega um conjunto de fatores, que representam o desempenho do conteúdo digital com um determinado critério estabelecido. Um exemplo destes fatores pode ser visto nas Tabelas 9.1, 9.2 e 9.3, retiradas de (Escudeiro, 2007).

Fatores	Domínio Técnico
F1	Objetos de aprendizagem
F2	Gestão de dados do utilizador
F3	Adaptabilidade
F4	Aspetos Técnicos
F5	Administração
F6	Gestão de conteúdos

Tabela 9.1: Características do Domínio técnico

Fatores	Domínio Ergonómico
F7	Ambiente de aprendizagem
F8	Usabilidade
F9	Arquitetura de ecrã
F10	Audio
F11	Interatividade
F12	Texto

Tabela 9.2: Características do domínio Ergonómico

Fatores	Domínio Pedagógico
F13	Cognitivo
F14	Afetivo
F15	Psicomotor

Tabela 9.3: Características do Domínio Pedagógico

9.2 QEF - Quantitative Evaluation Framework

Os modelos de avaliação devem adotar algumas normas como a ISSO/IEC 9126, padrão internacional para avaliação de *software* (esta norma não fornece requisitos para o *software*, mas define um modelo de qualidade que é aplicável a todos os tipos de *software*, avaliada num espaço de três dimensões) (Escudeiro et al., 2010). Como foi dito no início deste capítulo, a *framework Quantitative Evaluation Framework* (QEF), desenvolvida especificamente para avaliação de *software* educativo, é a aproximação que se pretende usar para avaliação do PEP.

O QEF é um instrumento para a avaliação quantitativa de *software* educacional, analisando o *software* educacional num espaço tridimensional ortogonal de qualidade. O QEF não se restringe apenas a medir a qualidade final de um sistema, mas, permite a avaliação da sua qualidade em qualquer momento durante o ciclo de vida, oferecendo liberdade na definição de critérios para os mais variados domínios aplicacionais, isto é, permite a adaptação a qualquer domínio e valência.

A qualidade de um dado sistema é medida em percentagem relativamente a um sistema hipoteticamente ideal cuja qualidade é presumida ser de 100%. O sistema é avaliado pela maior ou menor distância entre aquilo que o conteúdo digital faz e o que seria suposto fazer, medida no "espaço de qualidade". Cada uma das respetivas dimensões agrega um conjunto de fatores relativamente aos quais interessa determinar o grau de desempenho do sistema em estudo. Os fatores são representados por um conjunto de critérios de qualidade definidos para o hipotético sistema ideal. Deve-se ter em conta qual o objetivo final e em função deste, definir o conjunto de requisitos (critérios) de qualidade do produto. Estes critérios deverão permitir que o *software* se torne mais ou menos adequado aos objetivos que se deseja alcançar num processo de ensino/aprendizagem. Cada requisito deve depois ser quantificado a fim de medirmos a qualidade para que possa ser interpretada por todos os intervenientes da avaliação. Resumindo, os critérios são os elementos caracterizadores de cada fator e os fatores os elementos caracterizadores de cada dimensão (Escudeiro, 2007; Escudeiro, 2010).

Tendo em vista a avaliação de *software* produzido usando o QEF, Rosa Reis (Reis, 2016), após aprofundada pesquisa na área da engenharia de *software*, selecionou vários critérios de qualidade segundo as dimensões técnico/funcional, pedagógico/didática e ergonómica, conforme Figura 9.1 (esses 128 critérios, estão detalhadamente definidos no anexo da tese de Rosa Reis). Como mencionado atrás, esta seleção pretende ajudar a equipa de avaliadores a escolher um conjunto de requisitos de qualidade para o produto a ser avaliado. As dimensões abrangem vários fatores (ou indicadores) que agrupam os critérios. Os fatores mais importantes, destacados pela autora são: Navegação, Interação, Comunicação, Conteúdos Didáticos, Aprendizagem, Socialização, Facilidade de uso, Flexibilidade, Consistência, Segurança, Acessibilidade.

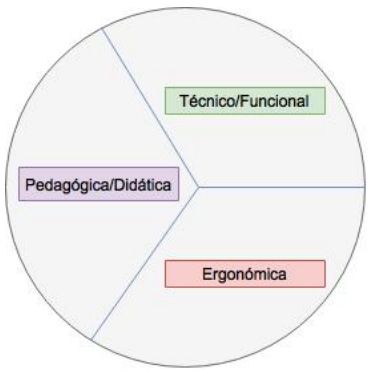


Figura 9.1: Dimensões para o QEF

O processo de medição da qualidade do sistema obtida desenvolve-se nos seguintes passos (Escudeiro, 2007):

1º – Classificação de critérios: Após identificação dos critérios, dos fatores e das dimensões, é necessário atribuir pesos aos respetivos critérios, ou seja, é necessário classificar os critérios de cada fator com um peso entre 1 e 10. A Figura 9.2 mostra uma possível escala conforme originalmente sugerida pelos autores do QEF.

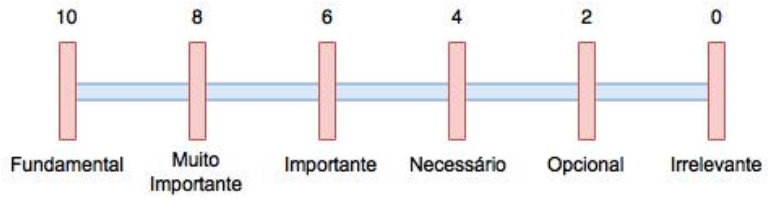


Figura 9.2: Escala de atribuição de pesos aos critérios de qualidade

2º – Classificação dos factores de cada dimensão: Para os fatores estes pesos tomam como referência o número de critérios do fator sobre o total de critérios da dimensão a que pertence. A contribuição do fator na dimensão é representada por um número real entre 0 e 1, indicando a relevância do fator na dimensão (numa dimensão o somatório dos pesos dos seus fatores terá de ser igul a 1).

3º – Avaliação de resultados: Após a identificação das dimensões, dos fatores, dos critérios e dos pesos de relevância de cada critério e fator, preenche-se uma matriz com os valores obtidos relativos a cada critério (em percentagem) que mede o cumprimento do sistema relativamente aos testes efetuados para cada critério. Uma vez completada a matriz, é feito o cálculo da qualidade do sistema, aplicando as formulas detalhadas em (Escudeiro, 2008).

4º – Cálculo do desempenho atingido por dimensão: Antes de obter um resultado global, é possível avaliar o desempenho do sistema em cada dimensão particular.

O objetivo de todo este processo é conseguir resultados quantitativos referentes ao produto a avaliar que sejam compreensíveis, aceitáveis e confiáveis pelos avaliadores. O valor obtido traduz percentualmente o grau de satisfação do sistema real avaliado relativamente ao propósito para o qual foi concebido.

9.3 Avaliação do PEP

Ao PEP (Plataforma de Ensino da Programação), descrito no capitulo 8, vai ser aplicado o QEF, modelo de avaliação apresentado anteriormente.

Após identificadas as dimensões, fatores e requisitos de qualidade para o PEP, e de acordo com a *framework* QEF é feito o agrupamento entre estes três elementos. Este agrupamento advém do facto desta *framework* avaliar o *software* num espaço tridimensional ortogonal, onde cada dimensão agrega um conjunto de fatores, aos quais interessa determinar o grau de desempenho do sistema em estudo. Os fatores são representados por um conjunto de critérios de qualidade definidos para o hipotético, sistema ideal representado no espaço de qualidade (Escudeiro, 2008). Na tabela 9.4 é possível observar as dimensões escolhidas para o PEP, agregadas aos respetivos fatores, os quais por sua vez estão agregados a um conjunto de critérios. Também foi feita a classificação de cada requisito numa escala de 0 a 100 obtendo assim a classificação final do PEP.

Dimensão Ergonómica		Cumprimento dos Requisito [0,100]	Q_j	Q_i	q
Fator	Requisito				
Navegação	O ambiente permite ao utilizador atuar e explorar de maneira natural?	90	73,6	72	
	O envolvimento do utilizador no ambiente é o mais simples possível?	80			
	O ambiente fornece instruções suficientes, formação e ajuda sobre o ambiente?	50			
	Uma vez que um utilizador encontra a ação correta na interface, o ambiente permite-lhe identificar qual o caminho correto para os efeitos que eles estão tentando produzir?	90			
	A informação apresentada é pertinente para a localização imediata dentro do ambiente?	70			
	Está claro para o utilizador qual a próxima ação a realizar?	90			
	O ambiente permite que o utilizador seja capaz de regressar ao ponto anterior sem se desorientar?	70			
	A navegação é intuitiva e facilmente memorizável?	90			
	O ambiente fornece informação ao utilizador sobre “onde estou”, “ onde quero ir? como faço para chegar lá?	90			

	O ambiente prevê um agente pedagógico com o propósito de guiar os utilizadores ao longo do sistema?	0			
	A navegação do sistema é transparente e permite o utilizador controlar as suas ações?	90			
Comunicação	São os canais de comunicação eficazes?	0	0		
	O ambiente permite que os utilizadores comuniquem uns com os outros através de vários ferramentas (chat, Vídeo,...)?	0			
Interação/ Interface	O ambiente permite que os alunos possam adquirir habilidades básicas e conhecimento sobre como utilizar o próprio ambiente?	70	85		
	O utilizador controla o comportamento do sistema)?	50			
	O ambiente prevê a possibilidade de sugestões de tarefas, missões e regras aos utilizadores?	60			
	O ambiente pode ser usado a longo prazo e permite que seja usado por utilizadores com diferentes perfis?	100			
	O envolvimento do utilizador com o sistema é suportado por vários tipos de ações?	70			
	O ambiente permite ao utilizador pesquisar e adquirir informação de forma fácil e intuitiva?	0			
	As ações efetuadas estão visíveis para os utilizadores?	20			
	A interface é compreensível?	90			
	O ambiente assegura que os botões com grafismos estejam definidos com texto?	100			
	O utilizador pode relembrar as regras das tarefas?	70			
Dimensão Pedagógica					
Fator	Requisito				
Conteúdos Didáticos	A informação existente no ambiente está consistente com o assunto abordado?	100	77,8	64	69%
	O conteúdo corresponde ao objetivo da aprendizagem e está de acordo com os perfis dos estudantes?	100			
	O professor pode participar e monitorizar as ações dos estudantes?	0			
	Os conteúdos estão relacionados com situações e problemas de interesse?	100			
	É dado ao aluno oportunidade de selecionar informações que lhe permitam relacionar informações que são importantes para o tipo de atividade que está envolvido?	0			
	O conteúdo está de acordo com o curriculum?	100			
	O conteúdo está de acordo com as caraterísticas dos estudantes?	100			
	O texto est legível, não contém erros ortográficos, e as frases estão bem construídas?	100			
	As tarefas são suficientemente complexas para que o aluno possa explorar os desafios, novas ideias e conhecimento?	100			
Aprendizagem	O ambiente “deixa vestígios” sobre as atividades realizadas pelos utilizadores, para mais tarde se poder refletir sobre elas?	60			

	O ambiente permite que os estudantes possam obter ajuda de forma a serem encorajados e motivados?	0	64,4		
	O ambiente permite que o professor ajude a criar as sequências de aprendizagem?	100			
	As tarefas estão projetadas de acordo com o grupo de utilizadores definidos?	100			
	O aluno é incentivado a adquirir competências?	50			
	O professor pode participar e monitorizar as ações dos estudantes?	0			
	As tarefas a executar seguem uma ordem pré-definida, ou seja, primeiro são apresentadas tarefas simples e, em seguida adicionadas elementos com maior complexidade, deforma a manter os alunos interessados no que estão a praticar?	100			
	Ambiente está adequado para os objetivos pedagógicos do professor?	100			
	Possibilita ao professor monitorizar se a aprendizagem está a ocorrer e como está ocorrendo?	70			
Socialização	O ambiente proporciona uma sensação de presença social de forma a facilitar a participação?	0	0		
	O ambiente suporta a aprendizagem colaborativa?	0			
Dimensão Técnico-Funcional					
Factor	Requisito				
Facilidade de uso	Os objetos ativos estão devidamente assinalados e explicados para promover a aprendizagem?	50	68,75	71	
	O ambiente impede que o utilizador cometa erros. Contudo se existirem erros o sistema deteta e avisa?	50			
	O ambiente permite que o utilizador seja capaz de aplicar o conhecimento adquirido e experiencia noutros ambientes?	50			
	O ambiente responde às ações do utilizador?	70			
	São as ações dos utilizadores visíveis e reconhecíveis?	70			
	Uma vez realizada a ação o ambiente permite aos utilizadores entender o feedback produzido?	70			
	Pode o utilizador efetivar a manipulação ou ação facilmente?	90			
	As respostas às ações são dadas em tempo suficiente?	100			
Flexibilidade	O ambiente permite a ligação a outros sistemas externos, tais como?	0	0		
Consistência/ Coerência	A resposta do sistema às ações do utilizador é previsível e informativa?	90	90		
	O utilizador é capaz de prever o resultado das suas ações?	90			
	O ambiente fornece respostas consistente às ações do utilizador?	90			
	São consistentes os nomes/informação e a estrutura?	100			
	A visualização da informação é consistente?	80			

Tabela 9.4: Tabela com as Dimensões, Fatores e Requisitos do PEP

9.4 Sumário

Neste capítulo procedeu-se à avaliação do primeiro protótipo da plataforma PEP seguindo a abordagem do QEF que foi aqui suscintamente apresentado.

Tendo-se considerado os pesos de todos os requisitos dos diferentes fatores das três dimensões todos iguais e após atribuirmos a cada um desses requisitos uma classificação numa escala de 0 a 100 que pareceu refletir honestamente o presente estado da ferramenta e utilizando uma folha de cálculo, obteve-se uma classificação por dimensão de 72%, 64% e 71%, o que se traduziu na avaliação final de 69%. Os resultados obtidos, para um primeiro protótipo, são bastante satisfatórios e encorajantes.

Mais do que avaliar quantitativamente a ferramenta no seu todo, pretendeu-se com este exercício identificar claramente as fragilidades a considerar para melhorar uma nova versão do PEP.

Entende-se que era bastante importante repetir este processo de avaliação com novas e mais apropriadas questões, recorrendo a professores e alunos como avaliadores. De facto, ao fazer a pontuaçãoo dos vários requisitos que foram mantidos de anteriores utilizações do sistema, detetou-se que alguns deles se classificavam naturalmente enquanto que outros não pareciam muito pertinentes.

10. CONCLUSÃO

Neste documento dissertou-se sobre a necessidade de encontrar formas de melhorar o processo de ensino/aprendizagem em cursos de *Introdução à Programação de Computadores*. Estudaram-se e discutiram-se métodos para atuar como fatores externos na motivação extrínseca e na autoconfiança dos alunos que facilmente se desinteressam quando enfrentam dificuldades. Nesse sentido propôs-se a combinação de duas aproximações baseadas em ferramentas computacionais já existentes mas que, na nossa crença, poderão quando interligadas resultar numa abordagem mais eficaz: a *animação de programas*, para ajudar a compreender como funcionam as soluções aos problemas propostos; e a *avaliação automática de programas*, para fornecer aos alunos *feedback* imediato sobre as soluções por eles desenvolvidas.

Foram, então, apresentadas duas abordagens que combinam de forma simétrica as duas técnicas referidas. A primeira delas já foi alvo de uma análise experimental em sala de aula, como se comenta a seguir; a segunda conduziu a implementação da Plataforma para Ensino da Programação, PEP, que poderá servir para ajudar o professor na aula ou o aluno em casa.

Em relação ao experimento realizado com alunos de Programação, pode dizer-se que a evolução do comportamento desses alunos ao longo da aula de duas horas mostrou que esta abordagem levou a um melhor desempenho por parte dos mesmos. Por um lado, notou-se

que o número de alunos com submissões aceites aumentou e, por outro lado, que o número de submissões aumentou e que o número de erros de compilação diminuiu. Isto significa que a motivação dos estudantes aumentou, enquanto que os erros de base foram reduzindo. Como foi sendo argumentado ao longo de todo o documento, a *motivação* foi uma das nossas principais preocupações. Os experimentos realizados também nos permitiram entender a melhor forma de conduzir futuras validações experimentais, como por exemplo, permitindo uma maior flexibilidade na gestão do tempo durante a aula assistida pela abordagem proposta.

Esses experimentos permitiram validar a hipótese de investigação enunciada na Introdução por recolha e análise dos fatores aí indicados para aferir que os alunos se motivaram e envolveram mais e consequentemente aumentaram o seu desempenho no domínio da aprendizagem da Programação.

A Plataforma de Programação de Ensino (PEP) proposta, prototipada por um grupo de alunos do Mestrado em Engenharia Informática, pode ser utilizada para ajudar o professor em sala de aula ou ajudar o aluno em casa, sendo este possivelmente o seu maior benefício. Essa plataforma, quando estiver totalmente implementada conforme protótipo atual e seguindo as sugestões de evolução/melhoria decorrentes do estudo feito com o avaliador de qualidade QEF, deve permitir medir os critérios de comprovação da tese aqui defendia, conforme apresentado como hipótese em 1.4.

10.1 Contributos

Consideram-se como contributos deste trabalho de doutoramento:

a) a identificação dos principais problemas que levam ao insucesso escolar nos cursos de Introdução à Programação;
b) a proposta de uma abordagem ao ensino da Programação baseada no princípio em que o aluno começa por ter de resolver um dado problema, devendo submeter a solução à avaliação automática para receber o respetivo *feedback*; depois procede à análise da solução correta com recurso à animação;
c) a proposta de uma segunda abordagem ao ensino da Programação baseada no princípio em que primeiro o aluno é exposto à análise dos problemas e sua resolução,

com o apoio da Animação passando depois a uma segunda fase onde, o próprio aluno, resolve problemas semelhantes, recorrendo à avaliação automática;

d) a definição de um esquema geral para testar em sala de aula a viabilidade e eficácia das abordagens ao ensino da Programação aqui propostas;
e) a especificação de uma Plataforma de Apoio ao Ensino da Programação baseada na segunda abordagem e a orientação do desenvolvimento de um primeiro protótipo, bem como a escolha de uma *framework* para avaliação da qualidade do *software* didático;
f) a mobilização de uma equipa interdisciplinar e interinstitucional que se criou e reuniu em torno deste trabalho de doutoramento para conceber uma proposta, intitulada "*Informa, Informática e Formação em Programação*", que foi submetida para financiamento à FCT na chamada de 2017 (a aguardar resposta).

O trabalho aqui apresentado e discutido, que conduziu aos contributos agora enumerados, foi alvo de validação pela comunidade científica tendo sido divulgado através das seguintes publicações:

1. P. Tavares, P.R. Henriques, E. Gomes (2017). A Computer Platform to Increase Motivation in Programming Students - PEP (CSEDU 2017). Porto, Portugal.
2. P. Tavares, P.R. Henriques, E. Gomes (2017). Motivação e Aprendizagem no Ensino da Programação – Sistema PEP.Técnicas para aumentar o envolvimento dos alunos. 12ª Conferência Ibérica de Sistemas e Tecnologias de Informação, 2017 (CISTI 2017), Portugal.
3. P. Tavares, P.R. Henriques, E. Gomes (2016). Computer-Supported Techniques to Increase Students Engagement in Programming. 8th International Conference on Computer Supported Education (CSEDU 2016). Rome, Italy.
4. P. Tavares, P.R. Henriques, E. Gomes, M.J.V. Pereira (2016). Técnicas para aumentar o Envolvimento dos Alunos na Aprendizagem da Programação. VII Congresso Mundial de Estilos de Aprendizagem (CMEA2016), Portugal.
5. P. Tavares, P.R. Henriques, E. Gomes (2015). Animation and Automatic Evaluation in Supporting the teaching of Programming. 10ª Conferência Ibérica de Sistemas e Tecnologias de Informação, Simpósio Doutoral (CISTI 2015), Portugal.

6. P. Tavares, P.R. Henriques, E. Gomes (2015). Animation and Automatic Evaluation to Support Programming Teaching". 7th International Conference on Computer Supported Education – Doctoral Consortium (CSEDU 2015), Portugal.

10.2 Direções para Trabalho Futuro

Em termos imediatos e na sequência direta deste projeto, é fundamental para concluir o que já foi feito e está previsto:

- Definir novas experiências em sala de aula que ajudem a continuar a aferir a eficácia das abordagens que foram propostas. Será importante experimentar as abordagens com alunos de cursos diferentes e tópicos distintos de complexidade variável. Será também relevante testar a proposta numa instância completa de uma Unidade Curricular de Programação;
- Melhorar a plataforma PEP introduzindo-lhe alguns dos artefactos de gamificação com vista a aumentar a motivação dos alunos de acordo com as ideias expostas no Capítulo 2, procedendo à sua reavaliação com o QEF;
- Procurar ferramentas de Animação e Avaliação Automática de Programas mais recentes que possam ser usadas como alternativa ao Jeliot e Mooshak acrescentando valor às abordagens propostas, nomeadamente oferecendo a capacidade de análise de código.

Para além destas tarefas diretas, outras direções de investigação daqui decorrentes podem ser propostas como grandes linhas de trabalho futuro:

1. Investigar formas de incluir instrumentos de avaliação em ferramentas de apoio à motivação como as estudadas.
2. Incluir estudos baseados em técnicas de análise de dados académicos (*Data Mining* ou outras) para a compreensão do insucesso escolar e para a fundamentação de futuras propostas de combate ao fracasso.
3. Estudar as influências que plataformas de detecção automática do perfil dos programadores podem trazer ao processo de ensino e avaliação em disciplinas de Programação.

11. REFERÊNCIAS

(ACM/IEEE, 2013) ACM/IEEE, “Computer Science Curricula 2013 -- Curriculum Guidelines for Undergraduate Degree Programs in Computer Science.” Final Report, 2013.

(Agrawal et al., 1994) Agrawal, R. and Srikant, R. “Fast Algorithms for Mining Association Rules”, In VLDB-94, 1994.

(Almeida, 2012) Almeida, D. “A motivação do Aluno no Ensino Superior: um estudo exploratório.” Dissertação apresentada ao programa de Mestrado em Educação da Universidade Estadual de Londrina, 2012.

(Almeida, 2013) Almeida, R. “Aprendendo Algoritmo com VisuAlg.” Editore Ciência Moderna Ltda., Rio de Janeiro (ISBN 978-85-399-0400-6), 2013.

(Almeida, 2016) Almeida, D. “Gamificação do Ensino da Programação num Curso Profissional da Área das Ciências Informáticas.” Tese de mestrado em Multimédia da Universidade do Porto, 2016.

(Awad, 2005) Awad, M. “A Comparison between Agile and Traditional Software Development Methodologies.” Submitted as partial fulfilment of the requirements for the Honours

Programme of the School of Computer Science and software Engineering, The University of Western Australia, 2005.

(Baldwin, 2015) Baldwin, D. “Getting Started with the Online Python Tutor CodeVisualizer.” work produced by OpenStax-CNX and licensed under the Creative Commons Attribution License 4.0, 2015.

(Bayardo et al., 1999) Bayardo, R. and Agrawal, R. “Mining the Most Interesting Rules.” In Proc. of the 5th International Conference on Knowledge Discovery and Data Mining, K DD-99, pp.145-153, 1999.

(Boas et al., 2013) Boas, I., Oliveira, N. and Henriques, P. “Agile development for education effectiveness improvement.” Proceedings of the XV international symposium on computers in education (SIIE'2013). Viseu, Portugal, 2013.

(Brusilovsky et al., 2004) Brusilovsky, P. and Spring, M. Adaptive, “Engaging, and Explanatory Visualization in a C Programming Course.” ED-MEDIA'2004 - World Conference on Educational Multimedia, Hypermedia and Telecommunications, 2004.

(Callahan, 2010) Callahan, M. “How Do I Motivate My Students?” Teaching Resources, Texas Tech University, 2010.

(Campos et al., 2011) Campos, E., Ramos, S. “Teorias da Motivação.” Dissertação do 2º Ciclo em Gestão de Recursos Humanos e Comportamento Organizacional (ISMT), pp. 125-142, 2011.

(Castro et al.,?) Castro, J.P., Pérez, M.Á., Regueras, L.M. and Verdú, M.J. “EduJudge System HandBook—How to Organize Programming Competitions in Moodle Courses.” ?

(Costacurta, 2008) Costacurta, A. P. “Animação de Algoritmos.” Graduação em Informática da universidade ederal do Panamá, Curitiba, 2008

(Cruz et al., 2008) Cruz, D., Henriques, P.R and Pereira, M.J.V. “Exploring and Visualizing the ‘alma’ of Documents.” in XATA08—Aplicações e Tecnologias Associadas, Évora, 2008.

(Cruz et al., 2009) Cruz,D., Béron, M., Henriques, P.R. and Pereira, M.J.V. “Code inspection approaches for program visualization.” Acta Electrotechnica et Informatica, 9(3):32–42, Jul-Sep 2009.

(Csikszentmihalyi, 1975) Csikszentmihalyi, M. “Beyond Boredom and Anxiety: Experiencing Flow in Work and Play.” San Francisco: Jossey-Bass. ISBN 0-87589-261-2, 1975.

(Csikszentmihalyi, 1990) Csikszentmihalyi, M. “Flow:The Psychology of Optimal Experience.” Harpers Perennial, NewYok, 1990.

(Csikszentmihalyi, 2014) Csikszentmihalyi, M. “Applications of Flow in Human Development and Education: The Collected Works of Mihaly Csikszentmihalyi.” Dordrecht: Springer, 2014. ISBN 978-94-017-9093-2, 2014.

(Elliott et al., 2015) Elliott, E., Fons, F. and Randell, A. “Business Architecture and Agile Methodologies.” Business Architecture Guild, February 2015.

(Escudeiro, 2007) Escudeiro, P. “Avaliação da Qualidade em conteúdos Digitais”, 2007.

(Escudeiro, 2008) Escudeiro, P. “Modelo de desenvolvimento de software educativo: uma proposta de implementação de um modelo de desenvolvimento de software educativo no sistema de Ensino Superior Português.” Tese de doutoramento, 2008.

(Escudeiro et al., 2010) Escudeiro, P., Bidarra, J. and Escudeiro, N. “Evaluating Educational Software.” in Systemics, Cybernetics and Informatics, Volume 8, n. 2, 2010.

(Espinoza et al., 2006) Espinoza, N. and Perdomo, B. “A Methodology for Educational Software Evaluation (ESE).” Emerging Trends and Challenges in Information Technology Management, Volume 1 and Volume 2 edited by Mehdi Khosrow-Pour, 2006.

(Esteves et al., 2003) Esteves, M. and Mendes, A.J. “OOP-Anim, a system to support learning of basic object oriented programming concepts.” CompSysTech’2003-International Conference on Computer Systems and Technologies, 2003.

(Esteves et al., 2004) Esteves, M. and Mendes, A.J. “A Similation Tool to Help Learning of Object Oriented Programming Basics.” 34th ASEE/IEEE Frontiers in Education Conference, pp. 20-23, October 2004.

(Faessler et al., 2006) Faessler, L., Hinterberger, H., Dahinden, M., Wyss, M. “Evaluating student motivation in constructivistic, problem-based introductory computer science courses.” ELEARN 2006 Proceedings, pp. 1178-1185, 2006.

(Fonte et al., 2014) Fonte, D., Boas, I., Oliveira, N., Cruz, D., Gançarski, A. and Henriques, P. “Partial Correctness and Continuous Integration in Computer Supported Education.” in Proceedings of the 6th International Conference on Computer Supported Education (CSEDU 2014), Volume 2. Barcelona, Spain, 2014.

(Gestwicki et al., 2004) Gestwicki, P.V. and Jayaraman, B. “JIVE: Java Interactive Visualization Environment.” OOPSLA’04, pp. 24-28, Vancouver, British Columbia, Canada, 2004.

(Godoi et al., 2009) Godoi, K. A. And Padovani, S. “Avaliação de material didático digital centrada no usuário: uma investigação de instrumentos passíveis de utilização por professores” in Production, Vol. 19, num. 3, pp.445-457, Setembro 2009.

(Gomes et al., 1999) Gomes, A. J. and Mendes, A.J. “A animação na aprendizagem de conceitos básicos de programação.” in Revista de Ensenanza y Tecnología, 1999.

(Gomes et al., 2004) A.J., Gomes, Mendes, A.J. and Marcelino, M.J. “Avaliação e Evolução de um Ambiente de Suporte à Aprendizagem da Programação.” VII Congresso Iberoamericano de Informática Educativa, 2004.

(Gomes, 2010) Gomes, A. J. “Dificuldades de aprendizagem de programação de computadores: contributos para a sua compreensão e resolução.” Dissertação submetida à Universidade de Coimbra para obtenção do grau de “Doutor em Engenharia Informática”, 2010.

(Guo, 2013) Guo, P.J. “Online Python Tutor: Embeddable Web-Based Program Visualization for CS Education.” SIGCSE’13, Denver, Colorado, USA Mar 6–9, 2013.

(Han et al., 2004) Han, J., Pei, J., Yin, Y. and Mao, R. “Mining Frequent Patterns without Candidate Generation: A Frequent Pattern Tree Approach.” Data Min. Knowl. Discov. 8, 1, January 2004.

(Hansen et al., 1999) Hansen, S.R., Narayanan, N.H. and Schrimpsher, D. “Helping Learners Visualize and Comprehend Algorithms.” Proceedings of the World Conference on Educational Multimedia, Hypermedia & Telecommunications (ED-MEDIA'99), 1999.

(Hansen et al., 1999) Hansen, S.R., Narayanan, N.H. and Schrimpsher, D. “Helping Learners Visualize and Comprehend Algorithms.” Proceedings of the World Conference on Educational Multimedia, Hypermedia & Telecommunications (ED-MEDIA'99), 1999.

(Heng et al., 2005) Heng, P., Joy, M., Boyatt, R. and Griffiths, N. ”Evaluation of the BOSS Online Submission and Assessment System”, 2005.

(Herzberg, 1968) Herzberg, F. “Teoria dos dois fatores.” Publicada na Harvard Business Review, 1968.

(Huang et al., 2013) Huang, W. and Soman, D. “A Practitioner´s Guide to Gamification of Education.” Reasearch Report Series Behavioural Economics in Action – Rotman School of Management – University of Toronto, 2013.

(Hughes et al., 2004) Hughes, C. and Buckley, J. “Evaluating Algorithm Animation for Concurrent Systems: AComprehension-Based Approach.” 16th Workshop of the Psychology of Programming Interest Group. Carlow, Ireland, In E. Dunican & T.R.G. Green (Eds), pp. 193-205, 2004.

(Hughes et al., 2004) Hughes, C. and Buckley, J. “Evaluating Algorithm Animation for Concurrent Systems: AComprehension-Based Approach.” 16th Workshop of the Psychology of Programming Interest Group. Carlow, Ireland. In E. Dunican & T.R.G. Green (Eds), Proc. PPIG 16, pp. 193-205, April 2004.

(Hundhausen et al, 2000) Hundhausen, C. and Douglas, S. “Using Visualizations to Learn Algorithms: Should Students Construct Their Own, or View an Expert’s?” Proceedings 2000 IEEE International Symposium on Visual Languages IEEE Computer Society Press, Los Alamitos, 2000.

(Hundhausen et al., 2002) Hundhausen, C., Douglas, S. and Stasko, J.T. “A Meta-Study of Algorithm Visualization Effectiveness.” in Journal ofVisualLanguages and Computing 13, pag 259-290, 2002.

(Joy et al., 2005) Joy, M., Griffiths, N., Boyatt, R. “The BOSS Online Submission and Assessment System.” in Journal on Educational Resources in Computing, Volume 5 Issue 3, September 2005.

(Kelleher et al., 2002) Kelleher, C., Cosgrove, D., Culyba, D., Forlines, C., Pratt, J. and Pausch, R. “Alice2: Programming Without Systax Errors.” in UIST’02, 2002.

(Kirby et al., 2009) Kirby, S., Toland, B. and Deegan, C. “Program Visualisation tool for teaching programming in C.” in Irish Learning Technology Association, July 2009.

(Klock et al., 2014) Klock, A., Carvalho, M., Rosa, B. and Gasparini, I. “CINTED – Novas Tecnologias na Educação.” Vol 12, nº2, 2014.

(Korhonen, 2003) Korhonen, A. “Visual Algorithm Simulation.” Dissertation for the degree of Doctor of Science in Technology at Helsinki University of Technology (Espoo, Finland), November, 2003.

(Leal et al., 2003) Leal, J.P. and Silva, F. "Mooshak: a Web-based multi-site programming contest system.” Software: Practice and Experience, vol. 33 n.6, pp. 567-581, 2003.

(Leal et al., 2008) Leal, J. and Silva, F. “Using Mooshak as a Competitive Learning Tool.” 2008.

(Leal et al., 2010) Leal, J.P. and Silva, F. “Using Mooshak as a Competitive Learning Tool” in Capítulo do livro.“ A New Learning Paradigm: Competition Supported by Technology”, pp. 91-105, 2010.

(Leal et al., 2011) Leal, E. A., Miranda, G. J. and Carmo, C. R. “Teoria da Auto Determinação: uma Análise da Motivação dos Estudantes do Curso de Ciências Contábeis.” III Encontro de Ensino e Pesquisa em Administração e Contabilidade, 2011.

(Lessa et al., 2011) Lessa, D., Czyz, J. and Jayaraman, B. “JIVE: A Pedagogic Tool for Visualizing the Execution of Java Programs.” SIGCSE Texas, USA, 2011.

(Manso et al., 2009) Manso, A., Oliveira, L.M.L and Marques, C.G. “Ambiente de Aprendizagem de Algoritmos – Portugol IDE. ”Actas da VI Conferência Internacional de TIC na Educação , Braga, Portugal, Janeiro 2009.

(Manso et al., 2015) Manso, A., Oliveira, L.M.L and Marques, C.G. “Portugol IDE – Uma ferramenta para o ensino de programação.” Fevereiro 2015.

(Maslow, 1970) Maslow, A. H. “Motivation and personality” (Rev. Ed.). New York: Harper & Row, 1970.

(McClelland, 1967) McClelland, D. “The achieving society.” New York: Free Press, 1967.

(McGregor, 1960) McGregor, D. “The Human side of entrepise.” New York: McGraw-Hill, 1960.

(Mendes et al., 2000) Mendes, A.J. and Gomes A.J. “ Suporte à aprendizagem da programação com ambiente SICAS.” V Congresso Ibero-Americano de Informática Educativa, December 2000.

(Monteiro, 2014) Monteiro, S., Santos, F. “Gamificação dos trabalhos de Grupo no Ensino Superior: o caso do Instituto Politécnico de Leiria.” Investigação, Práticas e Contextos em Educação, 2014.

(Oliveira et al., 2009) Oliveira, N., Pereira, M.J.V., Cruz, D. and Henriques, P.R. “Visualization of Domain-Specific Programs' Behavior.” Proceedings of VISSOFT'09 --- 5th IEEE International Workshop on Visualizing Software for Understanding and Analysis, Edmonton, Canada, pp. 37-40, 2009.

(Pereira et al., 1999) Pereira, M.J.V. and Henriques, P.R. “Animação de Algoritmos tornada Sistemática.” In 1º Workshop Computação Gráfica, Multimédia e Ensino. Leiria, 1999.

(Pereira, 2002) Pereira, M.J.V. “Sistematização da Animação de Programas.” Dissertação submetida à Universidade do Minho para obtenção do grau de doutor em Informática, ramo Tecnologia da Programação, Dezembro 2002.

(Proulx, 2000) Proulx, V. “Programming patterns and design patterns in the introductory computer science course.” in Proceedings of the thirty-first SIGCSE technical symposium on Computer science education, pp.80-84, New York, 2000.

(Queirós et al., 2012) Queirós, R. and Leal, J. “Exercises Evaluation Systems - An Interoperability Survey.” in Proceedings of the 4th International Conference on Computer Supported Education (CSEDU), Volume 1, pp.83-90, Porto, 2012.

(Queirós et al., 2015) Queirós, R. and Leal, J. “Ensemble: An Innovative Approach to Practice Computer Programming.” in R. Queirós (Ed.), Innovative Teaching Strategies and New Learning Paradigms in Computer Programming, pp. 173-201, Hershey, 2015.

(Rajala et al., 2007) Rajala, T., Jussi, M., Erkki, L. and Salakoski, K. “VILLE -A Language-Independent Program Visualization Tool.” Seventh Baltic Sea Conference on Computing Education Research (Koli Calling 2007), Koli National Park, Finland, November, pp.15-18, 2007.

(Rajala et al., 2008) Rajala¸ T., Laakso, M., Kaila, E.n and Tapio, S. ”Effectiveness of Program Visualization: A Case Study with the ViLLE Tool.” in Journal of Information Technology Education: Volume 7, Innovations in Practice, 2008.

(Ramos, 2013) Ramos, S. “Motivação Académica dos Alunos do Ensino Superior.” Psicologia.pt, O portal dos Psicólogos, 2013.

(Reis, 2016) Reis, S. R. “Modelação de Mundos Virtuais 3D-Análise Comparativa e Avaliação da Qualidade de Mundos Virtuais” Tese de Doutoramento submetida à Universidade de Trás-os-Montes e Alto Douro, 2016.

(Ribeiro, 2012) Ribeiro, A. “Jogo sério Colaborativo para Ensino da Programação a Crianças.” Tese de mestrado em Engenharia Informática e Computação, Faculdade de Engenharia do Porto, Fevereiro 2012.

(Ryan et al., 2000) Ryan, R., Deci, E. “Self-Determination Theory and the Facilitation of Intrinsic Motivation, Social Development, and Well-Being.” American Psychologist, Vol. 55, pp. 68-78, 2000

(Sankpal, ?) Sankpal, S. “Automatic Evaluation of Programming Assignments.” Master of Technology Programme in Computer Science and Enggineering, Indian Institute of Technology, Bombay, ?.

(Santos et al., 2006) Santos, R.P. and Costa, H.A.X. “Análise de metodologias e ambientes de ensino para algoritmos, estruturas de dados e programação aos iniciantes em computação e informática.” INFOCOMP: Journal of Computer Science, Lavras, v. 5, n. 1, pp. 41-50, 2006.

(Santos et al., 2010) Santos, A., Gomes, A. and Mendes, A.J. “Integrating New Technologies and Existing Tools to Promote Programming Learning.” in Journal of Algorithms, April 2010.

(Saraiya, 2002) Saraiya, P. “Effective Features of Algorithm Visualizations.” Thesis submitted to the Faculty of the Virginia Polytechnic Institute and State University for the degree of Master of Science in Computer Science, July, 2002.

(Silva et al., 2009) Silva, M.H., Emery, R.A., Neto, J.S. and Bezerra, Y.M. ”Estruturas de Programação: um Experimento com Jeliot.” IX Jornadas de Ensino Pesquisa e Extensão da UFRPE, 2009.

(Silva et al., 2010) Silva, M., Wendt, G. and Argimon, I. “A teoria da autodeterminação e as influências socioculturais sobre a identidade.” Psicologia em Revista, volume 16. ISSN 1677-1168, 2010.

(Silva et al., 2014) Silva, T., Mascarenhas, I., Medeiros, C. and Sousa, E. “A Motivação no Ensino Superior: Um Estudo com Alunos dos Cursos de Administração e Direito.” Journal of Management Analysis, Volume 3, pp.104-113.Fortaleza, 2014.

(Silva, 2002) Silva, C. R. “MAEP: Um Método Ergopedagógico Interativo de Avaliação para Produtos Educacionais Informatizados.” Tese de Doutoramento Submetida à Universidade Federal de Santa Catarina, Maio 2002.

(Silva, 2006) Silva, T. “G-Portugol- Manual da versão v1.0”. Abril 2006.

(Simões et al., 2013) Simões, J., Redondo, R.,Vilas, A. and Aguiar, A. “Proposta de Modelo de Referência para Aplicação de Gamificação em Ambientes de Aprendizagem Social.” Challenges 2013: Aprender a qualquer hora e em qualquer lugar, learning anityme anywhere, 2013.

(Simões, 2015) Simões, J. “Gamificação: Estratégias de Jogos Aplicadas ao e-Learning – Gamificação e Psicologia.” TecMinho e-Learning, 2015.

(Sorva et al., 2013) Sorva, J., Karavirta. V. and Malmi, L. “A Review of Generic Program Visualization Systems for Introductory Programming Education.” ACM Transactions on Computing Education (TOCE), Volume 13 Issue 4, November 2013.

(Stasko et al., 1996) Stasko, J. T., Kehoe, C. M. “Using Animations to Learn about Algorithms: An Ethnographic Case Study.” Technical Report GIT-GVU-96-20, September 1996.

(Stasko, 1990) Stasko, J. T. “TANGO: A Framework and System for Algorithm Animation.” IEEE Computer, Vol. 23, No. 9, pp. 27-39, Setembro 1990.

(Stolee, 2010) Stolee, K. “Kodu Language and Grammar Specification” August 2010.

(Taylor, 2011) Taylor, L., Parsons, J. “Improving Student Engagement.” Current Issues in Education. Vol.14, nº1. Retrieved from http://cie.asu.edu/, 2011.

(Tekdal, 2013) Tekdal, M. “The Effect of an Example-Based Dynamic Program Visualization Environment on Students’ Programming Skills.” in Educational Technology & Society, 16 (3), pp.400–410, 2013.

(Valentim et al., 2016) Valentim, E., Souza, R., Araújo, R., Oliveira, V. “Interatividade e Gamificação no Ensino Superior: Uma Análise em Fóruns de Discussão.” Internet Latent Corpus Journal, Vol. 6, nº1, pp. 150-172, 2016.

(Verdú et al., 2006) Verdú, E., Regueras, L.M., Verdú, M.J., Pérez, M.A. and Castro, J.P. “Improving the Higher Education through Technology-based Active Methodologies: A Case Study”, WSEAS Transactions on Advances in Engineering Education, vol. 3, no. 7, pp. 649–656, 2006.

(Verdú et al., 2011) Verdú, E., Regueras, L. M., Verdú, M. J., Leal, J. P., Castro, J. P. and Queirós, R. “A distributed system for learning programming online.”, Computers & Education 58, pp. 1–10, 2011.

(Virtanen et al., 2005) Virtanen, A. T., Lahtinen, E., and Jarvinen, H.M. “VIP, a Visual Interpreter for Learning Introductory Programming with C++.” in Proceedings of the Fifth Koli Calling Conference on Computer Science Education, Turku Centre for Computer Science, pp.125–130, 2005.

(Vroom, 1964) Vroom, V. “Work and motivation.” New York: Wiley, 1964.

(Webber et al., 2009) Webber, C., Boff, E. and Bono, F. “Ferramenta Especialiasta para Avaliação de Software Educacional.” XX Simpósio Brasileiro de Informática na Educação, 2009.

(Williams et al., 2011) Williams, K., Williams, C. “Five Key Ingredients for Improving Student Motivation.” Research in Higher Education Journal, pp. 104-122, 2011.

(Xavier et al., 2004) Xavier, G.M.C, Denise Ferreira Garcia, D.F., Silva, G.F. and Andréa Cristiane dos Santos, A.C.S. “Estudo dos Fatores que Influenciam a Aprendizagem Introdutória de Programação.” 2004