Observer e Concrete Factory aplicados na arquitetura de um laboratório virtual de aprendizagem desenvolvido em Unit 3D

Caio A. M. Campos
caio.a.campos@ufv.br
Universidade Federal de Viçosa
Florestal, MG, BR

Gláucia Braga e Silva
glaucia@ufv.br
Universidade Federal de Viçosa
Florestal, MG, BR

RESUMO

A utilização de padrões de projeto no desenvolvimento de jogos tem espaço na indústria e tem sido discutida na literatura, em especial, pelos ganhos obtidos em termos de um desenvolvimento rápido, mais modular e reutilizável. Este trabalho propõe a aplicação dos padrões *Observer* e *Concrete Factory*, com o intuito de se avaliar os impactos nos atributos de qualidade desempenho, modularidade e manutenibilidade em um fragmento da arquitetura padrão do UFV Virtual Labs. O UFV Virtual Labs é um laboratório virtual que simula diversas práticas laboratoriais desenvolvido utilizando tecnologias de jogos na engine gráfica Unity3D. Esses padrões foram inseridos em um fragmento isolado da arquitetura do laboratório, com o intuito de se melhorar, respectivamente, a gestão dos objetos gráficos manipulados e o controle dos gatilhos disparados pelas ações do usuário. Para se avaliar o impacto da aplicação dos referidos padrões, foram utilizadas métricas para se avaliar os atributos de qualidade selecionados, comparando os resultados antes e depois da inserção dos padrões.

ACM Reference Format:
Caio A. M. Campos and Gláucia Braga e Silva. 2022. Observer e Concrete Factory aplicados na arquitetura de um laboratório virtual de aprendizagem desenvolvido em Unit 3D. In *SBCARS '22:*. ACM, New York, NY, USA, 8 pages. https://doi.org/10.1145/nnnnnnn.nnnnnnn

1 INTRODUÇÃO

Ferramentas computacionais de apoio à educação têm sido utilizadas como um recurso importante no auxílio ao processo de aprendizagem. Em especial, ganham destaque aquelas que oferecem recursos lúdicos e interativos combinados com uma metodologia de aprendizagem ativa, como os Laboratórios Virtuais de Aprendizagem (LVAs). O LVAs podem ser desenvolvidos com base na tecnologia de jogos digitais, aumentando-se o engajamento e a motivação do estudante, e tornando-o protagonista do processo de ensino e aprendizagem.

Desde 2018, vem sendo desenvolvido na UFV-campus Florestal, um LVA dessa natureza. O UFV Virtual Labs trata-se de um laboratório virual de aprendizagem que simula práticas laboratoriais que trabalham temas associados à biologia molecular, tais como a presença de DNA em amostras biológicas, extração e purificação de DNA, amplificação de fragmentos de DNA pela técnica da PCR (Reação em Cadeia de Polimerase) e análise genômica de sequências de DNA. As práticas permitem a imersão do estudante no universo de um laboratório real de análises moleculares, incluindo o manuseio de equipamentos, vidrarias e reagentes. laboratório virtual supervisionado para práticas simuladas de genética e biologia molecular. Atualmente, o UFV Virtual Labs simula 10 práticas na área da biologia molecular, mas vislumbra-se além do aumento do número de práticas, extensões futuras para atender outras áreas do conhecimento, como computação, matemática, quimica e física.

Considerando-se o desenvolvimento com a tecnologia de jogos, o UFV Virtual Labs foi desenvolvido utilizando a engine gráfica Unity 3D e a linguagem orientada a objetos C# para programação da lógica das práticas simuladas. Do ponto de vista da arquitetura, alguns problemas de desempenho e alto acoplamento das classes devem ser investigados, em especial avaliando-se o potencial de escalabilidade do ambiente em relação ao número de usuários (estudantes), outras práticas e até mesmo outras áreas do conhecimento.

No que diz respeito ao desempenho, devido ao número excessivo de instâncias à objetos gráficos, tem-se um baixo FPS, o que impacta negativamente na experiência do usuário. Outro desafio está na gerência das ações realizadas pelo usuário durante a execução de uma prática sendo custoso o processo de incluir os possíveis gatilhos e erros que o estudante pode cometer. Como o usuário é livre para realizar o que deseja no ambiente de simulação, o controle dos gatilhos realizados pelas ações do usuário precisam estar constantemente sendo observados. Essa é uma tarefa complicada visto que, para adicionar uma nova ação ou um novo gatilho de evento, a arquitetura do sistema deve ser adaptada para validar a alteração.

Do ponto de vista da melhoria da qualidade em termos das questões arquiteturais mencionadas, observou-se que o uso de padrões de projeto tem sido uma solução praticada e reportada na literatura [12]. Nesse contexto, diversos padrões de design têm sido utilizados, com ou sem adaptações, na arquitetura dos jogos, no que compete à melhoria de atributos de qualidade. O uso de padrões criacionais no contexto de jogos tem sido discutido na literatura, uma vez que simplificam a manipulação de objetos gráficos. [6] utilizou de um padrão criacional adaptado para a Unity, o Concrete Factory, e observou ganhos em termos de reutilização do código em outros contextos similares. Padrões comportamentais da familia GoF, como o Observer, também ganham destaque, uma vez que jogos envolvem objetos sendo observados por observadores devido às interações com missões ou diversos agentes de gatilhos(Wu Ren, 2012).

SBCARS, 2022, Uberlândia, MG

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Este trabalho objetiva aplicar os padrões Concrete Factory e Observer à arquitetura do UFV Virtual Labs, com o intuito de avaliar o impacto em termos de desempenho, modularidade e manutenibiliadde do código. O Concrete Factory será aplicado como o intuito de se melhorar o desempenho da aplicação e o Observer para atuar numa melhor gerência de ações realizadas pelo usuário.

Na seção 2, serão apresentados os padrões de design bem como a importância deles nos atributos de qualiadde no contexto de jogos. Na seção 3 será feito o levantamento teórico de trabalhos correlatos que impulcionaram o desenvolvimento deste trabalho. Na seção 4, será abordada a metodologia da pesquisa juntamente com os objetos de estudo. Em seguida, na seção 5 será relatado a implementação dos padrões bem como a discussão dos resultados obtidos. Por fim, será sintetizado os resultados obtidos com esse trabalho bem como o levantamento de trabalhos futuros.

2 LABORATÓRIOS VIRTUAIS DE APRENDIZAGEM

Laboratórios Virtuais de Aprendizagem (LVAs) oferecem simulações virtuais de experimentos, até então realizados presencialmente, em um ambiente controlado que favorece o armazenamento, recuperação e troca de informações [18]. Estes funcionam muito bem como um complemento às aulas teóricas e práticas compensando a falta de recursos. De acordo com Amaral et al. [1], os LVAs possuem um importante papel na educação promovendo acessibilidade aos estudantes ou organizações que não possuem recursos necessários para os experimentos. Os LVAs constituem ainda importantes ferramentas de apoio ao ensino a distância, em situações de isolamento social, pandemias e outras que impeçam a continuidade das atividades escolares de forma presencial.

2.1 LVAs como jogos

Para tornar o processo de ensino e aprendizagem ainda mais inovador e criativo, aumentando o engajamento e a motivação do estudante, os jogos podem ser utilizados como uma ferramenta de ensino [4]. Desse modo, pode-se integrar tecnologias de jogos a LVAs tornando a aprendizagem mais eficiente. Os jogos educacionais promovem a aprendizagem ativa por meio de atividades motivadoras, prazerosas e desafiadoras. Cox e Bittencourt [10] afirmam que os jogos educacionais possuem uma grande importância cognitiva por desenvolverem atributos importantes durante o processo de aprendizagem, tais como atenção, resolução de problemas, tomada de decisões, colaboração, senso crítico e criatividade.

Segundo Schell [24], citado por Machado et al. [17], para a concepção de um jogo deve-se considerar 4 elementos principais:

(1) Estética, referindo-se à aparência e à interface gráfica, com impacto direto na retenção do usuário, na forma com que ele se interessa pelo jogo e em como as informações educacionais estão dispostas no mesmo.
(2) Mecânica, referindo-se à forma com a qual o usuário interage com os elementos do jogo para cumprir os objetivos. Segundo Cox e Bittencourt [10], os jogos com foco na aprendizagem devem voltar a sua atenção em como relacionar a interface e a mecânica do jogo de forma a potencializar a estrutura cognitiva do jogador.
(3) Narrativa, responsável pela história aderente ao jogo que fará com que o usuário fique entretido, tendo importante papel na retenção dos estudantes. Nela, serão apresentados o enredo e o roteiro da simulação. O enredo contém os personagens envolvidos, integrando as falas e histórias dos *Non-players Characters* (NPCs) e do próprio jogador, a motivação, os desafios que o jogador será exposto e quais caminhos ele poderá seguir para cumprir estes desafios. Já o roteiro, está ligado às etapas que o jogador deverá realizar para comprir a simulação.
(4) Tecnologia, que trata dos recursos tecnológicos requeridos, com destaque para os motores gráficos e frameworks que sustentam a construção desses ambientes. A escolha de uma tecnologia específica depende do tipo de plataforma que será aplicada, da complexidade do LVA e do nível dos recursos exigidos.

3 PADRÕES DE DESIGN E ATRIBUTOS DE QUALIDADE NO CONTEXTO DE JOGOS

Os padrões de design são aplicáveis aos diversoos tipos de sistemas orientados a objetos com o intuito de contribuir com a melhoria de um ou mais atributos de qualidade. Esta seção visa contextualizar o uso de padrões de design no contexto de jogos, frente aos atributos de qualidade que serão o foco de estudo deste trabalho.

3.1 Padrões de design aplicáveis ao contexto de jogos

Os padrões GoF (*Gang of Four*) são um conjunto de 23 padrões de projetos organizados em 3 categorias: criacionais, comportamentais e estruturais [13]. A implementação de um padrão específico, ou conjunto de padrões, visa impactar positivamente nos atributos de qualidade [12]. De acordo com [2], os padrões de projetos são aplicaveis em dois cenários no contexto de jogos: na mecânica e na programação dos jogos. A mecânica diz respeito ao conteúdo gráfico e estético do jogo, atrelado à usabilidade, ou seja, como e o que o usuário pode fazer na aplicação. Já a programação, diz respeito à lógica do jogo e em como o sistema se comportará frente aos gatilhos do usuário.

Apesar dos benefícios, o uso de padrões pode ter impactos diferentes com base no nível de conhecimento dos desenvolvedores. Isto é, as métricas de modularidade manutenibiliadde ou legibilidade podem ser comprometidas pelo maior número de clases e complexidade de código ao utilizar um padrão de projeto [16].

As categorias dos padrões GoF dizem respeito ao impacto do padrão no sistema em termos da manipulação dos objetos (gráficos ou não), da interação entre eles e da estrutura usada para representá-los. Este trabalho aborda o uso de padrões criacionais, chamados Factory, responsáveis pela criação e controle de objetos, e compartamentais, como o Observer, responsável pela interação de objetos com o comportamento de observadores e sujeitos observáveis.

3.1.1 Concrete Factory. O Concrete Factory, família dos Factories dos padrões GoF, é um padrão criacional cujo objetivo é realizar a manipulação de objetos, criando-os e gerenciando suas instâncias

[6]. As classes responsáveis por este padrão fornecem às demais a referência única dos objetos criados. A Figura 1 mostra a modelagem padrão do Concrete Factory.

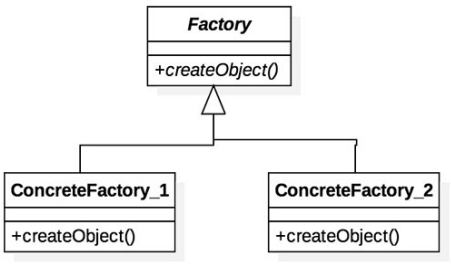


Figura 1: Modelagem padrão do Concrete Factory.[8]

No contexto da Unity3D, este padrão é aplicado na gerência de instâncias dos objetos gráficos, que são fisicamente manipulados no UFV Virtual Labs. Dessa forma, os factories criados impactam diretamente na memória alocada desses objetos, resolvendo problemas de instâncias duplicadas e gerência de memória de objetos ociosos.

O Concrete Factory pode ser estendido ao controle de objetos ociosos. Nesse caso, ele assume, também, a responsabilidade de destruir ou construir objetos sob-demanda e, a partir do momento que determinado objeto não está sendo mais utilizado ou terá pouca utilização futura, ele é destruído melhorando ainda mais o desempenho em memória.

3.1.2 Observer. O Observer é um padrão de projeto comportamental, dentre os 23 padrões GoF, que objetiva a reutilização de código e a simplificação das interações entre os objetos [21]. O padrão permite que se defina um mecanismo de assinatura para notificar múltiplos objetos (observadores) sobre quaisquer eventos que aconteçam com o objeto que eles estão observando (sujeito) [16]. Ou seja, as classes de sujeito precisam notificar aos observadores de alguma alteração interna, sendo esta a relação destaque do padrão. As classes de sujeito neste padrão são responsáveis por criar e remover observadores bem como os notificar das alterações internas, quando ocorrem. Já as classes observadoras são passívas na comunicação recebendo as informações notificadas e atualizando os respectivos atributos internos [21].

A Figura 1 mostra a implementação padrão do Observer.

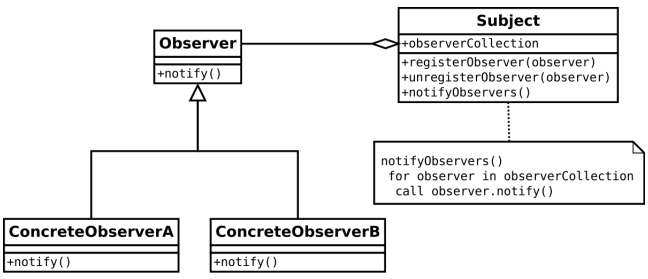


Figura 2: Modelagem padrão do Observer.[9]

3.2 Atributos de Qualidade e Arquitetura da aplicação

Para se garantir a qualidade de um sistema, são definidas métricas que possam mensurar os atributos de qualidade da aplicação. As métricas de software permitem quantificar atributos de qualidade em processos, produtos e projetos de software [25], segundo os requisitos não funcionais previamente estabelecidos [7].

Dentre os atributos de qualidade relevantes para o contexto do desenvolvimento de jogos, este trabalho destaca: o desempenho, uma vez que demanda-se por execuções cada vez mais rápidas, do ponto de vista gráfico, mas também logicamente eficientes; a modularidade, permitindo o desacoplamento e a reusabilidade do código; e a manutenibilidade, de forma a garantir maior facilidade nas mudanças, extensões e adições de novas funcionalidades ao jogo.

3.2.1 Métricas de Software. A modularidade de um software pode ser mensurada na facilidade de se replicar um modelo a situações parecidas sem a necessidade de grandes mudanças no código [16]. A manutenibilidade, se verifica pela facilidade em se dar suporte a um sistema. Esses dois atributos contribuem, também, para a inserção de novas funcionalidades sem a necessidade de realizar grandes alterações. De acordo com [3], esses atributos são extraídos analizando a combinação de acoplamento, herança, polimorfismo, coesão e tamanho de código.

Para analisar a modularidade e a manutenibilidade de código, o acoplamento entre as classes é um ponto importante a ser mensurado. A métrica *Couplng Between Classes (CBO)* foi desenvolvida para mensurar o relacionamento direto entre as classes [16]. Desse modo, quanto menor o acoplamento, melhor a modularização dos objetos e a facilidade de manutenibilidade [11]. A CBO conta o número de outras classes acopladas a uma classe específica.

Outra forma de mensurar o acoplamento entre as classes é por meio das métricas *Fan-In* e o *Fan-Out*, métricas estruturais cujo objetivo é medir a relação entre as classes. Essas métricas contabilizam a quantidade de acessos a uma função específica, sendo o *Fan-In* determinado pelo número de chamadas feitas a uma função (chamadas recebidas), e o Fan-Out, o número de chamadas feitas por uma função (chamadas realizadas) [19]. De forma similar ao CBO, valores altos de *Fan-In* ou *Fan-Out* indicam maior acoplamento, logo menor possibilidade de reuso entre as classes.

Por fim, uma métrica muito utilizada para mensurar a manutenibilidade de um sistema é a profundidade da árvore de herança (Depth of Inheritance Tree, DIT) de uma classe. Quanto maior seu valor, maior a profundidade e, por consequência, maior o acoplamento do sistema aumentando sua complexidade [11].

Para se mensurar o desempenho de um jogo, utiliza-se, comumente, o custo da taxa de quadros, apresentado ao usuário na forma de *frames per second* (FPS). Essa métrica diz respeito à quantidade de quadros, ou imagens, que são exibidas por segundo [22]. Para se determinar o custo, vários fatores são levados em consideração como o hardware que o jogo será executado e a complexidade lógica bem como os artifícios lógicos de otimização que foram empregados. Desta forma, o desempenho de um jogo é afetado pela quantidade de recursos como memória ou processamento que ele exige em diferentes qualidades de execução. Quanto maior a taxa de quadros (FPS), melhor é a exibição para o usuário. Em contrapartida, quanto

menor a taxa de quadros, maior será a percepção das imagens, dando a impressão de que o sistema está travando.

4 TRABALHOS RELACIONADOS

No contexto de avaliação de padrões de projeto em jogos digitais, o trabalho de [12] apresenta os 23 padrões do GOF exemplificando cada padrão aplicado em um contexto de jogos. O trabalho apresenta a modelagem comparativa, antes e após a utilização, bem como o foco de melhoria mediante aplicação de cada padrão. O estudo comparou um jogo específico desenvolvido com e sem padrões de projetos concluindo que a utilização dos padrões auxiliou na modularidade e teve impactos positivos na taxa de erros e depuração de código.

A pesquisa de [2] apresenta um estudo em cima de jogos open source desenvolvidos em Java mensurando o número de defeitos, taxa de depuração e o desempenho com base nos padrões de projetos utilizados. Os padrões Abstract Factory, Singleton, Composite, Adapter, Observer, State, Strategy, Template Method, Decorator, Prototype e Proxy foram escolhidos para serem observados e comparados. Ao final, o estudo relatou a relação positiva na utilização dos padrões Abstract Factory, Singleton, Composite, Observer, State, Strategy, Prototype e Proxy com o número de defeitos em jogos. Contudo, houve uma relação negativa com o uso dos padrões Adapter e Template Method. O trabalho relatou, por outro lado, o aumento na dependência entre as classes do Observer fazendo o desenvolvedor analisar com cautela quanto à utilização do padrão. Além disso, concluiu que o padrão Factory mostrou-se benéfico no que compete à reusabildiade. Já [5] apresentam um estudo apontando o impacto da aplicação dos padrões de projeto na manutenibilidade de jogos, sendo que 11 desses padrões se mostraram mais impactantes. De forma complementar, [20] aponta os padrões Abstract Factory, Observer, Decorator, Composite e Visitor como os que mais auxiliam na manutenibilidade e, por consequência, na extensibilidade do sistema.

A proposição de padrões de projetos específicos para uma tecnologia são frequentes no desenvolvimento de jogos visto que as arquiteturas de frameworks e engines gráficas geram dependências particulares. [6] propõe a utilização de 3 padrões no desenvolvimento de jogos utilizando a ferramenta Unity: Game Scene Controller (GSC), Concrete Factory e padrões de interação. O trabalho introduz os padrões no contexto da própria Unity, referenciando e adequando-os às dependências da ferramenta. Apesar de [6] analisar o Concrete Factory em termos de modularidade e extensibilidade, este trabalho irá propor uma análise de desempenho.

A Tabela 1 apresenta um resumo dos trabalhos relacionados, os padrões de projetos aplicados e o impacto, positivo(P) ou negativo(N), nos atributos de qualidade abordados neste trabalho: Modularidade(Mod), Desempenho(Des) e Manutenibilidade(Man).

Tabela 1: Tabela de padrões de projetos e os impactos nos atributos de qualidade.

Tabela dos Padrões			
Padrão	Man.	Des.	Mod.
Abstract Factory	P[20]	-	P[2]
Composite	-	-	P[20]
Decorator	N[2]	-	P[20]
Facade	-	P[23]	-
Chain of Responsability	N[15]	-	
Command	-	N[23]	-
Observer	P[20][2]	-	P[2]
Satate Strategy	-	-	N[15]
Visitor	P[20]	-	-

5 METODOLOGIA DA PESQUISA

A partir da revisão da literatura acerca de padrões de design aplicáveis ao desenvolvimento de jogos em Unity3D, foram selecionados os padrões Concrete Factory e Oserver para serem aplicados à arquitetura do UFV Virtual Labs, com o intuito de melhorar a qualidade em termos dos atributos de qualidade deste trabalho. Essa escolha também foi guiada pela análise de algumas questões arquiteturais da aplicação relacionadas a dois cenários: gestão de objetos ociosos (criação e destruição); e captura de diversos gatilhos acionados pelas ações do usuário. No que compete à gestão dos objetos ociosos, foram analisados os padrões Abstract Factory e Factory Method, dentre os 23 padrões GoF [14]. Ao explorar a aplicação desses padrões criacionais no contexto do desenvolvimento de jogos, identificou-se o Concrete Factory, uma adaptação da família Factory para o contexto da Unity. Já no que diz respeito à captura de gatilhos acionados pelas ações do usuário, a partir do estudo dos trabalhos relacionados, selecionou-se o Observer como padrão compartamental aplicável.

Os padrões selecionados foram então aplicados a cada um dos cenários identificados, de forma isolada, para que o impacto da aplicação pudesse ser medido de forma independente.

Dentre as práticas simuladas do UFV Virtual Labs, foi feito um recorte para uma prática específica de extração de DNA que abrangia passos específicos de manipulação de objetos que evidenciassem o uso do Concrete Factory. Para a aplicação do Observer, foi feito introduzido momentos que engatilhassem a interação do NPC com o personagem provocado por passos errados, além do próprio sistema de interfaces que apresentem na tela os passos atuais ao usuário.

Para observar o impacto de forma indepentende dos padrões, dado um mesmo contexto que consiga contemplar ambos os problemas, os padrões foram modelados individualmente e, em seguida, implementados no sistema. Desta forma eles poderiam ser analisados em individual, sem que um padrão interfira no resultado de outro.

Tendo em vista o contexto da aplicação, o Observer foi utilizado para analisar os ganhos em termos dos atributos de modularidade e manutenibilidade. Já o Concrete Factory, foi aplicado para se analisar possíveis ganhos em termos de desempenho. Para se analisar os resultados, foram aplicadas as métricas DIT, CBO e Fan-In/Fan-Out, para se avaliar a aplicação do padrão Observer; e FPS e gastos de memória, tanto para a lógica quanto para o sistema no geral, para se avaliar o uso do Concrete Factory. Para fins de comparações, as métricas foram aplicadas em dois cenários, antes e após o uso dos padrões.

6 APLICAÇÃO DOS PADRÕES OBSERVER E CONCRETE FACTORY NA ARQUITETURA DO UFV VIRTUAL LABS

Esta seção apresenta a aplicação dos padrões Concrete Factory e Oserver à arquitetura do UFV Virtual Labs, bem como o resultado da aplicação das métricas para se avaliar os possíveis ganhos em termos dos atributos de qualidade.

6.1 O UFV Virtual Labs

O UFV Virtual Labs é um simulador de práticas laboratoriais que consiste no agrupamento de diversas práticas em um contexto fechado e protocolado. Nele, os estudantes devem realizar um conjunto de passos mapeados para chegar a um resultado esperado. Contudo, possibilitando o desenvolvimento crítico e construtivo dos estudantes, as ações dos usuários dentro do ambiente virtual, apesar de roteirizadas, podem fugir ao esperado. Assim, ao final da execução da prática, a aplicação gera um relatório de desempenho apontando ao estudante seu progresso em uma determinada prática.

Assim, dois termos importantes foram modelados no domínio da aplicação: a *Prática Esperada* e a *Prática Realizada*. A *Prática Esperada* abrange um roteiro sequencial de atividades divididas em passos, que devem ser cumpridos para se atingir o resultado correto. Já a *Prática Realizada* abrange as atividades e os passos correspondentes executados pelo estudante durante a execução da prática simulada. Cada passo consiste num par de ação, verbo que designa um gatilho do sistema (pegar, andar, soltar...), e um objeto afetado pela ação, previamente modelado (pipeta, microtubo, solução...). A captura das ações realizadas é feita pelas entidades de controle e atreladas a um passo, dentro de uma ação. Após são comparadas com as ações esperadas e, caso não estejam compatíveis, um erro é apontado e é verificado se a ação errada é suficiente para cumprir o passo esperado ou não. Caso seja suficiente, ele prosseguirá na simulação, mesmo com o erro, caso não, ele deverá continuar tentando.

Para auxiliar o estudante a cumprir corretamente a prática, um roteiro é disponibilizado na forma de missões. Estas são apresentadas na tela, passo a passo, além de contar com a ajuda de um personagem fictício (non-player character, NPC), que explica algum passo novo ou auxilía quanto aos erros cometidos. A Figura 3 mostra a interface lateral com o passo atual que o usuário deve fazer, e a intervenções do NPC ao ver um novo gatilho de ação.

Para este trabalho, a prática a ser simulada é um recorte da prática de extração de DNA, que conta com passos simples para evidenciar os padrões aplicados. Nesse recorte, serão exploradas as capturas de ações pelo usuário, interferência dos NPCs frente aos erros e a gestão de instâncias de objetos gráficos.

6.2 Aplicação do Observer

Para se aplicar o Observer, um recorte da arquitetura do UFV Virtual Labs foi extraído no contexto de verificar as interações de gatilho do usuário dentro da prática abordada neste trabalho. A Figura 4 mostra as classes envolvidas na extração e utilização das interações do usuário, tanto para atualizar os passos realizados quanto para validar possível intervenção do NPC.

A classe *PraticaRealizada* armazena todas as interações realizadas pelo estudante, capturadas pela classe ControladoPraticaRealizada.

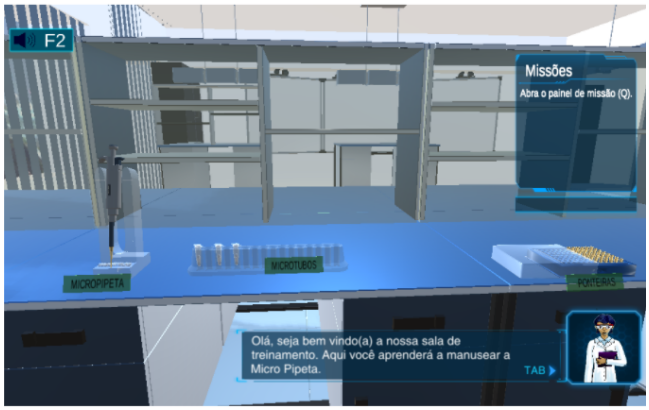


Figura 3: Interfaces dos passos e NPC do UFV Virtual Labs.

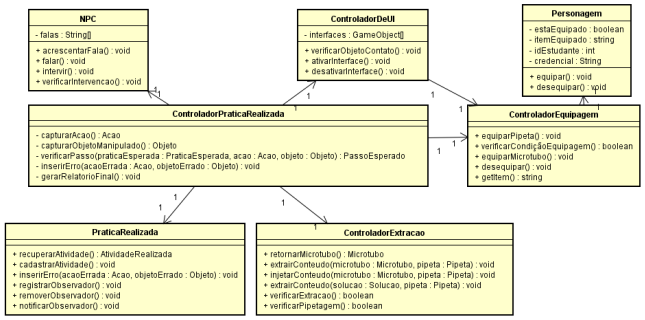


Figura 4: Recorte da arquitetura do UFV Virtual Labs para as classes responsáveis pelas ações realizadas pelo estudante

Essa controladora é responsável por recuperar e informar todos os passos realizados além de notificar ao NPC caso algum evento, cujo gatilho está mapeado, aconteça para que possa ser feita alguma intervenção.

O Observer foi inserido nesse contexto entregando a responsabilidade de notificar frente à alterações dos passos à classe PráticaRealizada eximindo de sua controladora tal responsabilidade. Além disso, as ações realizadas pelo usuário, abstraídas pela classe Personagem, são notificadas, com o padrão, pela mesma e não mais pela controladora da prática. Desta forma, tanto o ControladorDeUI, responsável por alterar as informações dos passos, quanto o NPC, que interfere em algumas ações engatilhadas pelo usuário, são ditos observadores dos sujeitos, controlador de prática e personagem.. A Figura 5 ilustra o diagrama de classes após a inserção do padrão Observer.

6.2.1 Avaliação do uso do padrão Observer. Após a aplicação do padrão, foram aplicadas as métricas CBO e DIT. A Tabela 2 mostra os resultados obtidos para as métricas antes e depois do padrão.

Como pode-se observar, a distribuição de responsabilidades contribuiu para a diminuição do valor do CBO na classe *ControladorPraticaRealizada* que, antes do padrão estava sobrecarregada. Desta forma foi atribuída à classe *Personagem*, a responsabilidade de notificar aos observadores qualquer interação engatilhada pelo usuário.

Tabela 2: CBO e DIT antes e após a implementação do Observer

Classe	CBO(Antes)	DIT(Antes)	CBO(Após)	DIT(Após)
ControladorPraticaRealizada	5	0	3	0
NPC	0	0	0	1
ControladorDeUI	0	0	0	1
PraticaRealizada	2	1	2	2
Personagem	0	0	1	1

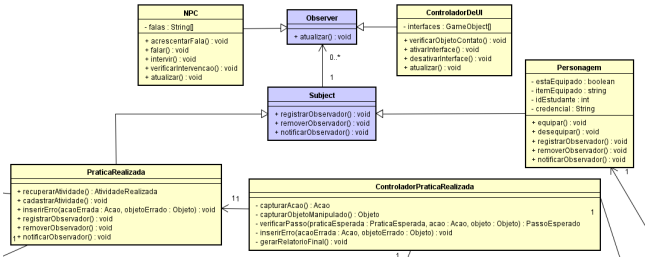


Figura 5: Inserção do padrão Observer no recorte da arquitetura padrão do UFV Virtual Lab.

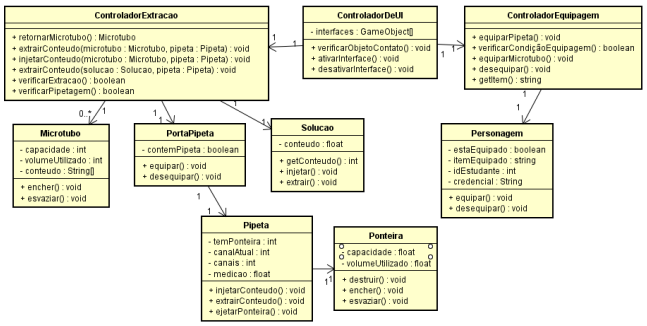


Figura 6: Recorte da arquitetura do UFV Virtual Labs para as classes responsáveis pela abstração dos objetos gráficos.

Contudo, a aplicação do padrão implica em aumentar a profundidade da árvore de herança dessas classes que, por sua vez, devem acatar às especificações do padrão. Por este motivo, houve aumento do valor de DIT, o que poderia tenoricamente causar uma piora na manutenibilidade. No entanto, destaca-se que, apesar do aumento, caso novos observadores e observados precisem ser usados, basta acrescentá-los ao modelo sem grandes esforços, como especializações das Classes Observer e Sujeito, respectivamente.

Outras métricas aplicadas para se avaliar a aplicação do Observer foram o Fan-In e o Fan-Out, conforme ilustrado na Tabela 3. Nela, são apresentados os resultados antes e após a aplicação do padrão. Ressalta-se que os valores representam o somatório dos valores parciais de Fan-In e Fan-Out nas funções explicitadas, visto que os métodos tabelados possuem tanto valores só para uma das métricas quanto para as duas e, o interesse é em mensurar o conjunto delas..

No geral, os valores totais dessas funções apresentaram significativa melhora após a distribuição das responsabilidades com o uso do padrão Observer. A melhora nesses valores tem relação com os valores obtidos com a CBO, em virtude dos ganhos em termos de desacoplamento das classes.

6.3 Aplicação do Concrete Factory

O diagrama de classe da Figura 6, apresenta as classes responsáveis pelos objetos gráficos implementados e que foram utilizados na pratica simulada, de extração de DNA. A classe *ControladorPraticaRealizada* se relaciona com as entidades *ControladorExtracao* e *ControladorDeUI*,presentes no diagrama da Figura 4, a fim de capturar as interações realizadas nestes objetos pelo usuário.

As classes de entidade representadas na Figura **??** são responsáveis por abstrair os objetos gráficos e as classes de controle, responsáveis por instanciá-los e controlar suas referências, além de informar as alterações realizadas nestes objetos pelas interações do usuário.

O padrão Concrete Factory foi aplicado neste recorte intermediando a criação dos objetos entre as classes de entidade e as controladoras. Dessa forma, foi possível desenvolver a criação e a destruição de instâncias e objetos gráficos sobre demanda, beneficiando a quantidade de objetos instanciados. A Figura 7 mostra a modelagem do padrão Concrete Factory no contexto apresentado.

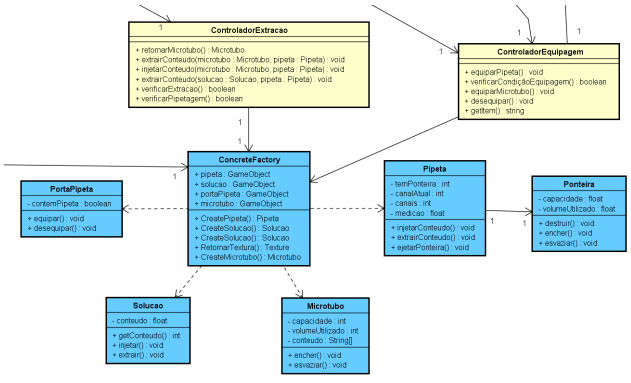


Figura 7: Modelagem do padrão Concrete Factory no contexto de criação de objetos.

Quando um objeto é requisitado por qualquer controladora, a fábrica *ConcreteFactory* entrega a única instância do objeto, evitando múltiplas instâncias de um mesmo objeto. Além disso, se um objeto não se faz mais necessário, a classe *ConcreteFactory* o destrói e mantém apenas os não ociosos.

Tabela 3: Fan-In/Fan-Out antes e após a implementação do Observer

Métodos	Sem Observer	Com Observer
`ControladorPratica.validarPasso()`	5	3
`PraticaRealizada.notificarObservador()`	-	1
`Personagem.Equipar()`	2	1
`NPC.falar()`	3	1
`ControladorDeUI.ativarInterface(obj)`	3	1
`ControladorPratica.notificarNPC()`	1	-

6.3.1 Avaliação do uso do padrão Concrete Factory. Após a implementação do padrão Concrete Factory, foram avaliadas as métricas de FPS e uso de memória. A análise de memória foi dividida entre: memória destinada à lógica do sistema e memória geral, que leva em consideração a renderização dos objetos gráficos. Os testes foram feitos com 5 repetições da prática, realizadas passo a passo pelo próprio sistema e os resultados correspondem à média dos valores de cada repetição. Para a análise de memória, foram considerados os 20 segundos iniciais de simulação, visto que é o período onde se observou a maior variedade de ações que engatilham o uso do Concrete Facotry. Já para a análise do FPS, foram observados os primeiros 10 segundos pelo mesmo motivo.

A Figura 8(a) confirma um ganho de memória com a implementação do padrão do Concrete Factory. Os ganhos estão relacionados, principalmente, pelo papel de facilitador na disponibilização de instâncias únicas. Pode-se observar também que no geral, o custo para a inicialização do sistema aumenta com o uso do padrão. Isso se dá pelo número adicional de classes usadas/requeridas pelo próprio padrão. No entanto, quando se pensa em escalar o sistema, esse custo adicional pode ser compensado.

A relação entre tempo e memoria utilizada pela lógica do sistema pode ser visualizada na Figura 8(b). Durante a maior parte da execução, o custo com o padrão é superior ao custo, em memória, sem o padrão, o que é justificável. O uso do padrão torna o sistema mais complexo pelo número adicional de classes e métodos para realizar o controle. Podemos observar também uma queda brusca de consumo de memória ao final da execução, no segundo 19. Este fato se dá pela destruição dos objetos não utilizáveis que acontece neste momento. Apesar do consumo de memória para a lógica, no geral, sernsuperior com o uso do padrão, podemos observar que este custo é atrelado a um ganho em memória quando se observa o gasto total deste recurso.

A Figura 9 mostra os resultados obtidos analizando o valor do FPS durante o tempo de simulação. Um ponto importante a se destacar é o ganho de FPS no início da simulação. Sem o padrão, todas as instâncias são criadas ao inicializar o sistema, sendo ou não necessárias naquele momento. Desta forma, o custo de inicialização é maior afetando a taxa de quadros. Contudo, ao longo da utilização do sistema observa-se que o ganho é ínfimo e no segundo 7, justamente quando ocorre a destruição e instanciação de objetos, a taxa de FPS cai. Contudo, observa-se que, no geral, o uso de padrão torna o FPS mais consistente e estável.

Apesar dos ganhos estarem em escalas pequenas, deve-se destacar que este é apenas um recorte do sistema e que, provavelmente, os ganhos possam ser mais significativos quando se considera a escalabilidade do sistema para novas práticas, com mais objetos em cena sendo manipulados por múltiplos usuários.

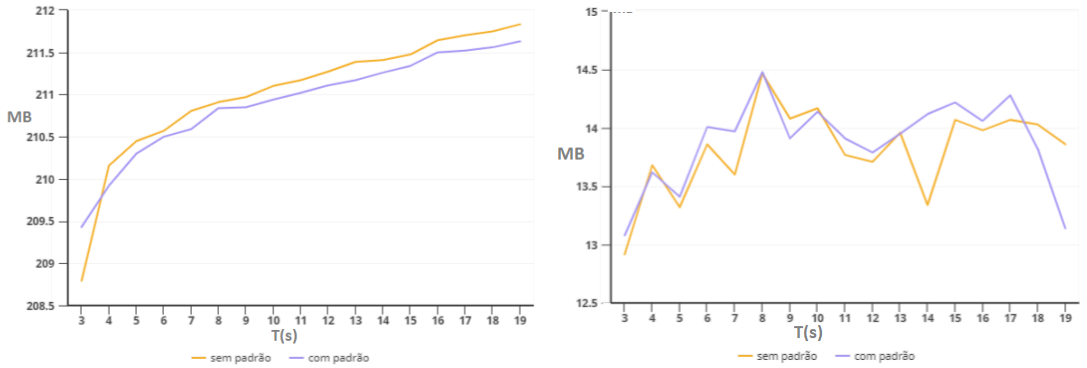
7 CONSIDERAÇÕES FINAIS

Este trabalho apresentou a aplicação dos padrões Observer e Concrete Factory à arquitetura do UFV Virtual Lab, um laboratório virtual de aprendizagem, com o intúito de se avaliar o impacto em termos de desempenho, modularidade e manutenibilidade. Os padrões foram selecionados com base nos principais problemas observados na aplicação. O Concrete Factory foi aplicado no contexto de criação de classes modelo, bem como a gerência dos objetos gráficos. Já o Observer foi inserido para controlar as capturas de gatilhos para as interfaces gráficas e para as interferências do NPC.

O Observer apresentou ganhos em termos de desacoplamento das classes, observadas pelas métricas CBO, *Fan-In* e *Fan-Out*. Contudo, verificou-se que a profundidade da árvore de heranças, medida pela métrica DIT, foi comprometida apresentando maiores valores para quase todas as classes envolvidas. Observa-se que esse aumento era esperado uma vez que a maioria dos padrões de projeto são modelados com relações polimórficas e/ou aquelas que envolvem generalizações e/ou uso de interfaces.

O Concrete Factory apresentou melhora no desempenho geral, apresentando um ganho médio superior à versão sem uso do padrão, consumindo menos memória total por justamente realizar a gerência das intâncias dos objetos. Por outro lado, a memória destinada à logica do sistema foi maior do que na versão sem o padrão. Isso se justifica pela maior complexidade gerada com a utilização do Concrete, em termos das classes próprias do padrão. Por fim, os ganhos no FPS se mostraram mais evidentes na inicialização do sistema se mostrando mais consistente e estável no decorrer do tempo. Apesar dos ganhos obtidos serem pouco significativos em termos práticos, ressalta-se que os padrões foram aplicados a apenas um pequeno fragmento do UFV Virtual Labs. Se escalarmos a aplicação para outras simulações maiores e mais completas, tendo em vista o número de usuários simultâneos e a somatória do custo atrelado, possivelmente os ganhos seriam mais evidentes.

Tendo em vista a escalabilidade do UFV Virtual Labs para atender à diversos usuários e ao aumento significativo do número de práticas, em trabalhos futuros sugere-se a quantificação dos resultados do Concrete Factory verificando o impacto no sistema geral. Além disso, propõe-se a aplicação de outros padrões já reconhecidos na literatura como relevantes para o desenvolimento de jogos e que se aplicariam ao contexto do UFV Virtual Labs, como State/Strategy, para controlar a mudança de estados dos objetos, em acordo ao Observer, e o Prototype, para melhorar a prototipação de objetos



(a) Resultados de Tempo(s)XMemória Alocada(MB) para o sistema, sem e com padrão.

(b) Resultados de Tempo(s)XMemória Alocada(MB) para a lógica, sem e com padrão.

Figura 8: Análise de gastos de Memória(MB) em relação ao tempo(s) de execução, sem e com o uso do Concrete Factory

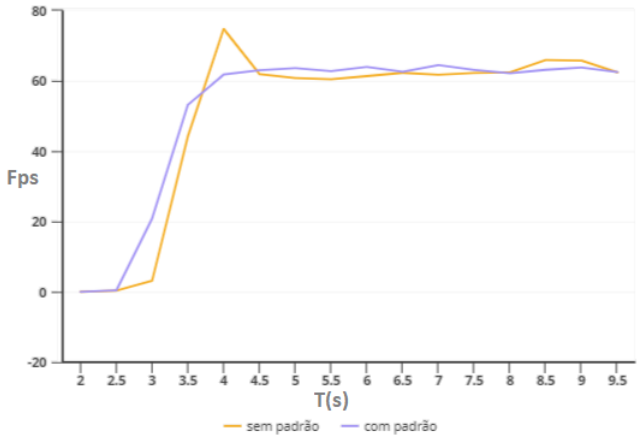


Figura 9: Análise do FPS antes e após a utilização do Concrete Factory.

gráficos com comportamento semelhantes, mas características particulares, sem a necessidade de se remodelá-los.

REFERÊNCIAS

[1] Érico Amaral, Bárbara Ávila, Herik Zednik, and Liane Tarouco. 2011. Laboratório virtual de aprendizagem: uma proposta taxonômica. *RENOTE-Revista Novas Tecnologias na Educação* 9, 2 (2011).

[2] Apostolos Ampatzoglou, Apostolos Kritikos, George Kakarontzas, and Ioannis Stamelos. 2011. An empirical investigation on the reusability of design patterns and software packages. *Journal of Systems and Software* 84, 12 (2011), 2265–2283.

[3] Jagdish Bansiya and Carl G. Davis. 2002. A hierarchical model for object-oriented design quality assessment. *IEEE Transactions on software engineering* 28, 1 (2002), 4–17.

[4] Pebertli Nils Alho Barata. 2015. *Framework para criação de laboratórios virtuais: diminuindo a lacuna entre teoria e prática em engenharia elétrica.* Ph. D. Dissertation. Universidade Federal do Pará.

[5] Staffan Björk and Jussi Holopainen. 2006. Games and design patterns. *The game design reader* (2006), 410–437.

[6] Nick Bucher. 2017. Introducing Design Patterns and Best Practices in Unity. In *Proceedings of the SouthEast Conference*. 243–247.

[7] Vera Lúcia Xavier de Sales Calçado. 2007. Influência da utilização de processo unificado, testes e métricas na qualidade de produtos de software. (2007).

[8] Kayun Chantarasathaporn and Chonawat Srisa-an. 2006. Energy conscious factory method design pattern for mobile devices with C# and intermediate language. In *Proceedings of the 3rd international conference on Mobile technology, applications & systems*. 29–es.

[9] W. Commons. [n. d.]. .

[10] Kenia Kodel Cox and Roberto Almeida Bittencourt. 2017. Estudo Bibliográfico sobre o Processo de Construção de Jogos Digitais: A necessidade de sinergia entre o educar e o divertir. *Revista Brasileira de Informática na Educação* 25, 01 (2017), 16.

[11] CURSO DE CIÊNCIAS DA COMPUTAÇÃO—BACHARELADO. [n. d.]. *TÍTULO: FERRAMENTA DE APOIO À COLETA DE MÉTRICAS EM SOFTWARE ORIENTADO A OBJETOS ÁREA: Engenharia de Software. Palavras-chave: Métricas de software orientado a objetos. Modelo CMMI.* Ph. D. Dissertation. UNIVERSIDADE REGIONAL DE BLUMENAU.

[12] Roberto Tenorio Figueiredo. 2014. *Padrões de Projeto GOF aplicados ao Desenvolvimento de Jogos Eletrônicos.* Master's thesis. Universidade Federal de Pernambuco.

[13] E Gamma, R Helm, et al. 2000. Design Patterns Ele m ent of Reusab le Object-O riented Softare.

[14] Erich Gamma, Richard Helm, Ralph Johnson, John Vlissides, and Design Patterns. 1995. *Elements of reusable object-oriented software.* Vol. 99. Addison-Wesley Reading, Massachusetts.

[15] Javier Garzás, Félix García, and Mario Piattini. 2009. Do rules and patterns affect design maintainability? *Journal of Computer Science and Technology* 24, 2 (2009), 262–272.

[16] Tova Linder. [n. d.]. How Does the Implementation of the Observer Pattern in Freecol Affect Reusability, Extendibility and Flexibility of the Application as measured by CBO? ([n. d.]).

[17] Liliane S Machado, Thaise Kelly de Lima Costa, and Ronei Marcos De Moraes. 2018. Multidisciplinaridade e o desenvolvimento de serious games e simuladores para educação em saúde. *Revista Observatório* 4, 4 (2018), 149–172.

[18] Roberto Correia de Melo and João Alberto Osso Jr. 2008. Laboratórios virtuais e ambientes colaborativos virtuais de ensino e de aprendizagem: conceitos e exemplos. *Revista de Informática Aplicada* 4, 2 (2008).

[19] Luís Felipe Garlet Millani. 2013. Análise de correlação entre métricas de qualidade de software e métricas físicas. (2013).

[20] Lutz Prechelt, Barbara Unger, Walter F. Tichy, Peter Brossler, and Lawrence G. Votta. 2001. A controlled experiment in maintenance: comparing design patterns to simpler solutions. *IEEE transactions on Software Engineering* 27, 12 (2001), 1134–1144.

[21] Wu Ren and Wenyun Zhao. 2012. An observer design-pattern detection technique. In *2012 IEEE international conference on computer science and automation engineering (CSAE)*, Vol. 3. IEEE, 544–547.

[22] Adriano Pereira REZENDE. 2013. Otimização de jogos para dispositivos móveis com Unity3D. (2013).

[23] Jakub Rudzki. 2004. How design patterns affect application performance–a case of a multi-tier J2EE application. In *International Workshop on Scientific Engineering of Distributed Java Applications*. Springer, 12–23.

[24] Jesse Schell. 2008. The Art of Game Design: A book of lenses. 2008. *Burlington, MA: Elsevier* (2008).

[25] Ian Sommerville. 2011. Software engineering 9th Edition. *ISBN-10* 137035152 (2011), 18.