**UNIVERSIDADE FEDERAL DO PAMPA**

**Pedro Henrique França Silva**

# Desenvolvendo um Jogo com Padrões de Interface para Ajudar no Aprendizado de Programação

Alegrete
2019

**Pedro Henrique França Silva**

# Desenvolvendo um Jogo com Padrões de Interface para Ajudar no Aprendizado de Programação

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Engenharia de Software.

Orientador: Prof. Me. Jean Felipe Patikowski Cheiran

Alegrete
2019

**Pedro Henrique França Silva**

# Desenvolvendo um Jogo com padrões de interface para ajudar no Aprendizado de Programação

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software da Universidade Federal do Pampa como requisito parcial para a obtenção do título de Bacharel em Engenharia de Software.

Trabalho de Conclusão de Curso defendido e aprovado em 27 de junho de 2019
Banca examinadora:

**Prof. Me. Jean Felipe Patikowski Cheiran**
Orientador
Unipampa

**Profa. Dra. Amanda Meincke Melo**
Unipampa

**Prof. Dr. Claudio Schepke**
Unipampa

## RESUMO

Padrões de interface são vastamente utilizados no desenvolvimento de software atualmente. No entanto, não existem muitos padrões de interface, destinados a jogos, documentados. Os que existem não estão estruturados de uma forma clara para o desenvolvedor que quer utilizá-los. Nesse sentido, o objetivo deste trabalho é coletar padrões de interface existentes, que podem ser aplicados em jogos, e estruturá-los em um *template* que possa ser compreendido e utilizado por desenvolvedores. Para isso, será criado uma prova de conceito em que um jogo será criado para que esses padrões possam ser investigados e aplicados. O tema escolhido para o jogo é o ensino de programação. Com isso, foram realizados estudos sobre como os jogos são usados na educação e quais foram as tentativas de ensinar programação com eles. Outro ponto estudado foram os padrões, sobre o qual foram verificadas as formas com que os padrões de interface são estruturados e outras tentativas parecidas de criar padrões de interface para jogos. A metodologia escolhida para se gerenciar o desenvolvimento do jogo e a estruturação dos padrões foi uma variação da metodologia ágil *Scrum*. Mudanças em algumas práticas do *Scrum* foram necessárias pela existência de apenas um membro no time *Scrum*. Nos resultados, um catálogo de padrões foi gerado. Além disso, uma avaliação do jogo foi feita com alunos e professores, em que foi verificada a efetividade do jogo no ensino de Programação e como os padrões afetaram a percepção dos envolvidos.



**Palavras-chave**: Padrões de Interface. Ensino de Programação. Jogos Educacionais.

**ABSTRACT**

Interface patterns are widely used in current software development. However, there are not many interface patterns for Games. Those that exist are not structured in a clear way for the developer who wants to use them. In this sense, the objective of this work is to collect interface patterns that can be applied in games, and structure them into a template that can be understood and used by developers. For this, a game will be created so that these patterns can be investigated and applied. The theme chosen for the game is programming teaching. With this, studies where analyzed on how games are used in education and what the attempts were to teach programming with them. The way the interface patterns were structured and how they were applied were also studied. The methodology chosen for managing the development of the game and structuring the patterns was a variation of the agile methodology Scrum. Changes in some Scrum were required by existence of only one member in the team. In the results, a catalog of Interface patterns was generated. In addition, an evaluation of the game was made with students and teachers, in which was measure the effectiveness of the game in the teaching of Programming and how patterns affected the perception of those involved.

**Key-words**: Interface patterns. Programming Teaching. Educational Games.

## LISTA DE FIGURAS

SUMÁRIO

# 1 INTRODUÇÃO

Jogos de computador estão se tornando uma forma proeminente de entretenimento (FANG, 2010). De acordo com Adams, MacNamara e Richard Wainess (2011), há uma grande evidência de que os jogos são populares, pois 60% da população Americana, acima de 6 anos, joga regularmente. Nesse sentido, muitos desenvolvedores estão escolhendo atuar na área de jogos. Para ajudar na criação do jogo, muitos deles podem utilizar padrões para resolver problemas que surgem durante o desenvolvimento. No entanto, a área de Padrões de Interface não é bem documentada na literatura. Dessa forma, há uma necessidade de criar um catálogo de Padrões de Interface que possa ajudar desenvolvedores na criação de interfaces de jogos. A criação de somente um catálogo de Padrões de Interface pode não ser expressivo o suficiente para que um novo desenvolvedor entenda como determinado Padrão funciona. Com isso, pode ser necessária a criação de uma Prova de Conceito que demonstre a aplicação desses Padrões de Interface. Nesse sentido, um jogo pode ser criado para demonstrar esses padrões. Para que a função do jogo criado não se limite a somente mostrar padrões, o jogo pode ser desenvolvido como uma ferramenta educativa.

Os jogos podem ser utilizados para o ensino de conteúdos variados. De acordo com Barendregt (2010) existem várias razões para se introduzir os jogos educativos na sala de aula. Um deles é que os jogos de computador podem motivar os alunos a aprender, a partir da vontade deles de jogar bem o jogo. Também, os jogos oferecem uma poderosa ferramenta de aprendizagem, providenciando interações e a possibilidade de 'aprender fazendo'. Para se criar um jogo educativo, deve-se definir um tema a ser ensinado. Com isso, um dos temas que alunos de computação têm dificuldade é a programação. Dessa forma, o jogo pode abordar o ensino dessa disciplina.

Em um mundo que está se tornando cada vez mais informatizado, a programação é uma habilidade muito útil e é base para a formação de vários profissionais na área da Computação. Nos anos recentes, a demanda por programadores e o interesse dos estudantes por programação vem aumentando rapidamente. No entanto, aprender programação é difícil. Novos programadores frequentemente têm dificuldades para desenvolver suas habilidades, fazendo com que cursos que envolvem programação tenham uma grande quantidade de desistências e geralmente sejam taxados como difíceis pelos alunos (ROBINS; ROUNTREE, 2003).

De acordo com Hernandez et al. (2010) há uma grande variedade de fatores que podem ser apontados para justificar as altas taxas de evasão que são muitas vezes observadas nos primeiros anos de cursos de Computação. Alguns destes fatores estão relacionados às expectativas dos calouros sobre o curso em si que está longe da realidade e a demanda dos cursos de Computação por conhecimentos sólidos em matemática. Além disso, outra causa de abandono pode estar relacionada ao ensino da base da lógica de programação, que pode exigir dos estudantes níveis de abstração mais elevados do que estão acostumados

a ter, além de consideráveis habilidades para a resolução de problemas em geral. Tarefas propostas por esses cursos normalmente são planejadas para fazer os alunos desenvolverem habilidades cognitivas que normalmente não tinham sido desenvolvidas antes. Nesse sentido, muitos alunos do curso de Engenharia de Software da Universidade Federal do Pampa(Unipampa) passam pelas dificuldades apresentadas. Dessa forma, há um grande número de desistências, fazendo com que a taxa de alunos que se forma seja baixa. Uma forma de facilitar o aprendizado desses alunos seria utilizar jogos de computador para aumentar o interesse e a facilidade de aprender, já que existem vários jogos educativos desenvolvidos.

## 1.1 Objetivos

O objetivo deste trabalho é catalogar padrões de interface para jogos através da busca dos padrões já existentes e aplicar esses padrões em jogo educativo que apoie o aprendizado de programação para mostrar seu funcionamento. Ao final do trabalho é esperado que tenhamos uma lista de padrões de interface, documentados de forma estruturada, direcionados a jogos, que possam ser aplicados por outros desenvolvedores.

Um objetivo secundário deste trabalho é que o jogo educativo criado realmente ajude no aprendizado dos alunos de Computação, sendo que os alunos da Unipampa poderão, também, ser beneficiados por ele.

## 1.2 Organização do trabalho

Primeiro é feita uma fundamentação teórica sobre os temas abordados nesse trabalho. Após isso, será tratado sobre trabalhos relacionados ao tema, expondo o que foi feito em cada um e analisando os resultados e as propostas de cada trabalho. Depois, iremos utilizar as informações encontrados nos trabalhos relacionados para propor o *design* de um jogo que atenda as necessidades do problema proposto. Depois, os padrões catalogados são descritos e os resultados da avaliação feita são mostrados. Por último, serão feitas considerações finais sobre este trabalho, no qual o autor retoma os assuntos abordados, além de descrever as limitações do trabalho e possíveis trabalhos futuros.

# 2 FUNDAMENTAÇÃO

Padrões de interface e desenvolvimento de jogos educativos são assuntos muito abrangentes. Para entendê-los foi necessário realizar um estudo. Nesta Seção são explicados os conceitos que foram estudados para a proposta do trabalho. Primeiro, são apresentados os problemas que alunos enfrentam para se aprender programação. Em seguida é apresentada a forma como jogos são aplicados na educação. Depois, é apresentado como jogos educativos são desenvolvidos. Posteriormente são discutidos os padrões de projeto em geral, além de explicar os padrões de interface.

## 2.1 Problemas para Aprender Programação

Em cursos de Computação e Informática uma das metas está definida em torno da capacidade do aluno apresentar soluções para diversas classes de problemas encontrados no cotidiano das pessoas, das organizações e de muitos outros elementos. Através de um programa busca-se um mecanismo para obter soluções para um conjunto de problemas ou processo. Dessa forma, instruções são fornecidas ao computador que as executa em uma ordem lógica formando assim uma solução para um problema (JÚNIOR; RAPKIEWICZ, 2004).

Aprender programação não é fácil, pois adquirir e desenvolver conhecimento sobre ela é um processo altamente complexo. A programação envolve muitas habilidades cognitivas e representações mentais sobre o problema a ser resolvido (ROGALSKI; SAMURÇAY, 1990). Muitas dessas habilidades cognitivas devem ser adquiridas durante o aprendizado de programação, pois os alunos, geralmente, não as adquiriram antes. Com isso, é estimado que se precise de 10 anos para se fazer um programador novato se tornar um especialista (ROBINS; ROUNTREE, 2003). Programação como um domínio de conhecimento se difere de outras áreas como Matemática ou Física de duas maneiras: a primeira é que os conceitos utilizados na programação não são vistos diariamente como, por exemplo, a recursão e variáveis (em contraste a conceitos como soma e velocidade que são comuns no cotidiano); e a segunda é que as operações ocorrem dentro de uma máquina e isso pode fazer com que o funcionamento do programa não seja transparente para os alunos (ROGALSKI; SAMURÇAY, 1990).

As metodologias de ensino de programação podem estar se tornando ineficazes, pois os textos de informática geralmente expõem os recursos de uma linguagem aos estudantes sem aplicação prática, sem um objetivo específico ou um desafio ao qual devam responder. Essa metodologia de ensino talvez fosse satisfatória quando os programas de computador apresentavam uma interface muito simples com o usuário, por meio de telas com informações apenas sob a forma de textos, e os objetivos em termos de programação eram muito limitados. No entanto, isso mudou. O aluno que estiver estudando sob essa metodologia não terá oportunidades de aprendizagem para desenvolver sua capacidade de compreensão e abstração do pensamento lógico-computacional, pois a literatura técnica

disponível emprega exemplos prontos, distantes de sua realidade. Ao ingressar no mercado de trabalho, em geral como estagiário de programação, esse aluno terá dificuldades em relacionar o conhecimento recebido com as necessidades reais da empresa (FONTES; SILVA, 2008). Com isso, uma forma de diminuir essa dificuldade é utilizar uma forma interativa de ensino. Essa forma de ensino pode ser feita através de jogos, que vem sendo usados há muito tempo para o ensino de assuntos variados.

## 2.2 Os Jogos na Educação

Existem várias maneiras de se definir um jogo. Autores costumam dar várias definições para o termo. De acordo com a autora McGONIGAL (2012), os jogos nunca apareceram em tantas formas como hoje. Atualmente, os jogos podem ser acessados de várias maneiras, através de consoles, computadores, celulares e etc. Esses jogos também vêm em vários formatos e podem ser destinados a um único jogador, a vários jogadores ao mesmo tempo ou a vários jogadores ao mesmo tempo através de uma rede online. Os jogos também têm uma duração variada, podendo durar de segundos a centenas de horas. Com toda essa variedade, definir um jogo é difícil. O autor Schell (2014) define os jogos em 10 características:

1. **Jogos são jogados voluntariamente:** Os jogadores têm que aceitar todas as regras e definições do jogo para que o jogo possa ocorrer;

2. **Jogos têm objetivos:** Os jogos têm objetivos que devem ser ultrapassados pelo jogador, mesmo aqueles que dependem da sorte;

3. **Jogos têm conflitos:** A maioria dos jogos passam a sensação de um conflito de forças. Mesmo nos jogos de um jogador, em que o conflito parece não estar presente. Como por exemplo o jogo tetris, em que o jogador está sempre lutando contra os cubos que estão sempre caindo e devem ser alocados na melhor posição possível;

4. **Jogos têm regras:** As regras impõem limitações em como os jogadores podem atingir os objetivos do jogo.

5. **Jogos podem levar a derrota ou vitória:** Muitos jogos têm um vencedor ao final do jogo e isso se torna um grande incetivo para o jogador, fazendo com que ele sempre desenvolva abordagens diferentes para o jogo, estimulando sua criatividade;

6. **Jogos são interativos:** Diferentes de outras formas de entretenimento, os jogos são interativos. Os jogadores tem controle de muitos aspectos do jogo e isso é delimitado pelas regras;

7. **Jogos têm desafios:** Os desafios impedem os jogadores de completarem os objetivos facilmente;

8. **Jogos podem criar valores internos próprios:** Os jogos podem ter uma lógica que não se aplica ao mundo real. Como por exemplo uma pontuação gerada através de um desafio ser utilizada para comprar um item qualquer.

9. **Jogos envolvem jogadores:** Os jogadores são os atores mais importantes de um jogo. Dessa forma não há sentido em existir um jogo que não tenha jogadores.

10. **Jogos são sistemas fechados formais:** Os jogos são sistemas, ou seja, todos os componentes do jogo se conectam e funcionam juntos, disponibilizando uma única experiência. Eles são fechados, pois há um limite para o sistema. E também são formais, ou seja, o sistema está claramente definido.

Apesar de jogos e vídeo games serem mais comumente utilizados para entretenimento, é importante entender que eles podem ser ferramentas de aprendizado extremamente poderosas. Eles permitem criar novas oportunidades de aprendizado e são importantes ferramentas para aqueles que cresceram utilizando computadores (PRENSKY, 2005). O potencial dos jogos para melhorar e apoiar a aprendizagem tem sido amplamente discutido. Frazer et al. (2014) apontam que jogos estão promovendo uma ampla gama de valores e habilidades que incluem resolução de problemas, tomada de decisão, a motivação, feedback em tempo real, assistência, aprendizado colaborativo, coleta de dados e análise, testes de hipóteses e desenvolvimento de habilidade de debate. Muitos estudos como Ebner e Holzinger (2007), Papastergiou (2009), Divjak e Tomić (2011), revelam que os jogos têm um impacto positivo na motivação e aprendizagem dos alunos. No entanto, para que o jogo ajude na aprendizagem é necessário que o *design* do jogo seja feito em uma estratégia que consiga aproveitar todos os benefícios que um jogo possa trazer para a educação.

## 2.3 Desenvolvimento de Jogos Educativos

De acordo com Crawford (1984), desenvolvimento de jogo é um processo artístico e também é um processo técnico. O *designer* de jogos persegue objetivos artísticos grandiosos, mesmo quando ele tem que se esforçar através das montanhas de código. O processo de desenvolver um jogo habita dois mundos muito diferentes: o mundo artístico e o mundo técnico. Primeiramente, para se iniciar o desenvolvimento de um jogo é necessário se estabelecer uma meta e um tópico. A meta é importante para que os desenvolvedores consigam tomar decisões baseadas nessas metas. Muitas vezes em um desenvolvimento de um jogo não é possível desenvolver tudo o que foi idealizado para ele. Dessa forma, com uma meta em mente os desenvolvedores podem tomar decisões com mais facilidade. Depois é necessário determinar o tópico, ou seja, do que o jogo vai tratar, onde a história vai se passar, etc. Após determinar a meta e o tópico, é necessário realizar uma pesquisa e a preparação para desenvolver o jogo. Nessa fase os desenvolvedores

devem pesquisar o tópico estabelecido para jogo, pois quanto mais rica for a história do jogo, mais o jogador se sentirá imerso em seu universo. Depois de realizar a pesquisa, é o momento de se iniciar a fase de *design* do jogo, em que será feito o projeto do jogo e será delimitado os requisitos, o escopo e a fronteira do jogo. Após completada a fase do *design*, se inicia a fase de desenvolvimento, em que será implementado tudo o que foi definido na fase anterior. Finalmente vem a fase de testes, em que todo o jogo será testado e será verificado qual as falhas presentes no jogo para que sejam corrigidas. O ideal é que o jogo seja testado por seus potencias consumidores.

Em contraponto ao processo abordado anteriormente, existem vários processos mais atuais para desenvolvimento de jogos. De acordo com Petrillo (2008), devido à natureza incremental que os jogos têm atualmente e ao aumento da complexidade de desenvolvimento, processos de desenvolvimento no formato de cascata não conseguem cobrir todos os problemas no desenvolvimento. Com isso, processos mais responsivos a mudanças deveriam ser adotados. Nesse sentido, ele propôs uma metodologia chamada *Game Agile Methods Applied* (GAMA) que se baseia em vários métodos ágeis de desenvolvimento. O gerenciamento do projeto é baseado no Scrum, com isso os ciclos de desenvolvimento devem ser curtos, interativos e incrementais. O GAMA é dividido na fase exploratória e fase de execução. Na fase exploratória são definidos os conceitos básicos do jogo e é formada pelas seguintes atividades:

1. **Elaborar conceito do jogo:** é a fase em que os principais elementos do jogo são definidos, como roteiros, os personagens e os ambientes. O documento gerado deve descrever apenas as linhas gerais de jogabilidade. Também são escolhidas a arquitetura e a tecnologia a ser utilizada.

2. **Determinar atores e estórias de usuário:** a partir do documento do jogo são definidos os atores e estórias de usuário que formam o jogo, indicando seus relacionamentos. Com isso, surge um diagrama de casos de uso.

3. **Elaborar estórias de usuários:** mapeados os casos de usos, eles são especificados em estórias de usuário, em cartões do tipo fichas pautadas, nos quais são definidas as ações, os fluxos, as interações e os comportamentos dos atores no jogo. Nesse processo, membros de toda a equipe devem participar, especialmente, os roteiristas e a equipe de arte. Dessa atividade podem surgir novos requisitos para complementar o digrama de casos de uso, além de protótipos de interface. Os casos de teste para as estórias também devem ser elaborados nessa fase.

4. **Planejar interações:** com as estórias detalhadas, é feita uma reunião de planejamento das iterações, na qual as estórias são priorizadas, além de classificadas segundo sua complexidade e risco. Essa atividade irá gerar o *backlog* do produto.

A fase de execução é formada pelas atividade de planejamento e execução de tarefas para a construção do jogo. Essa fase é formada pelas seguintes atividades e seus respectivos produtos:

1. **Definir e estimar tarefas:** as estórias priorizadas são selecionadas para a formação do *sprint*. Para cada estória, são determinadas as tarefas a serem realizadas para a implementação, tanto artística quanto de programação. As tarefas são escritas em cartões de tarefas e estimadas em pontos, utilizando-se a técnica de *planning poker*. Um *backlog* do *sprint* e cartões de tarefas devem ser produzidos nessa fase.

2. **Reunião de pé:** cada dia é iniciado com uma reunião rápida e de pé, na qual todo o integrante da equipe relata o que fez no dia anterior e o que pretende realizar. Ao final, os participantes assumem as tarefas, realocando-as no quadro e atualizando o gráfico de *burndown*. Um quadro de tarefas e um gráfico de *burndown* são esperados ao final dessa tarefa.

3. **Sessão de modelagem ágil:** caso necessário, na sessão de modelagem ágil são feitos os diagramas e esboços para a execução das tarefas.

4. **Sessão de implementação:** durante a implementação das tarefas são aplicadas as práticas de desenvolvimento guiado por teste, refatoração contínua, tanto de código-fonte quanto de arte, som ou outros artefatos do jogo.

5. **Executar atividades de qualidade:** após a integração dos módulos, o resultado do software deve ser testado por uma equipe especializada, realizando-se testes de caixa preta que atendam aos casos de teste definidos para as estórias; testes de cobertura; teste de integração; e, caso necessário, testes de carga. Os defeitos e aperfeiçoamentos podem ser relatados em uma ferramenta de *bug tracking* e lançados como novas tarefas no próprio sprint ou no seguinte.

6. **Integrar versão:** se todas as estórias implementadas atendem aos casos de teste ou atingiu-se o prazo final do *sprint*, o jogo é integrado em uma versão para demonstração.

7. **Reunião de retrospectiva:** ao final do *sprint* é realizada uma reunião, com toda a equipe, objetivando refletir sobre o processo de trabalho realizado, propondo aperfeiçoamentos para a próxima iteração.

8. **Entregar a versão final:** a última atividade consiste em empacotar a versão final para a distribuição, seja através de mídias ou pela Internet.

Realizar o *design* de um jogo educativo, adiciona mais um fator ao *design* de um jogo: o processo de ensino. De acordo com Moreno-Ger et al. (2008), existem três formas

de se utilizar jogos na educação. A primeira delas é fazer o jogo baseado no conteúdo, ou seja, primeiro é pensado o conteúdo que o jogo terá e depois disso é adicionada a jogabilidade e a diversão em cima desse conteúdo. Essa técnica não é efetiva, pois quando o aspecto de entretenimento falha, a maioria das vantagens do aprendizado baseado em jogos são perdidas. A outra técnica é a utilização de jogos já existentes para educar alunos. Alguns jogos são muito ricos em informações que podem ser usadas para o ensino. A grande vantagem dessa técnica é o custo. No entanto nem sempre o conteúdo desses jogos são coerentes com o que se quer ensinar, pois eles são desenvolvidos com foco no entretenimento e algumas informações podem ser mudadas para deixar o jogo mais interessante. Nenhuma das técnicas anteriores são ideais, pois focam nos dois extremos de diversão e aprendizado. Dessa forma, o ideal é uma técnica que encontre um equilíbrio entre as duas anteriores. Ou seja, desenvolver um jogo que tenha uma grande base educacional e que também seja divertido. No entanto, esse balanço não é fácil de se alcançar, apesar de existirem alguns jogos de sucesso utilizando essa técnica como o jogo Virtual Leader, que ensina os jogadores sobre lideranças. O jogo proposto neste trabalho irá utilizar a terceira abordagem, equilibrando os dois extremos para alcançar um bom resultado. Além da abordagem ao design do jogo, outro fator que pode ajudar a criar um bom jogo educativo é a utilização de padrões de interface.

## 2.4 Padrões de Projeto

Muitas vezes desenvolvedores se deparam com um problema e se perguntam: será que alguém já enfrentou esse mesmo problema e o resolveu? Provavelmente o problema enfrentado por esse desenvolvedor já foi resolvido e existe um mecanismo que documenta esses problemas e suas soluções encontradas. De acordo com Gamma et al. (1994), cada padrão descreve um problema que ocorre ao longo do tempo no nosso meio ambiente e, em seguida, descreve o núcleo da solução para esse problema, de tal forma que se pode usar esta solução um milhão de vezes mais, sem nunca fazê-lo da mesma forma duas vezes. Coplien e Alexander (1996) define os objetivos de um padrão:

- **Aumentar a produtividade:** Com o conhecimento de padrões, o retrabalho é evitado, pois as soluções descritas já foram utilizadas várias vezes. Além disso, projetos que utilizaram padrões são mais fáceis de se descobrir o funcionamento. Com isso, a manutenção se torna muito mais fácil;

- **Reduzir o Intervalo de Desenvolvimento:** Padrões são uma espécie de reuso de projetos já existentes. Com isso, eles reduzem o tempo necessário para se montar uma solução;

- **Reduzir o Custo:** Através dos objetivos anteriores, o tempo e o esforço envolvido no desenvolvimento irão reduzir. Com isso, o custo total do desenvolvimento

também irá reduzir;

- **Aumentar a Satisfação do Cliente:** Esse objetivo é atingido quando os outros objetivos listados são cumpridos.

Outra ideia que Coplien e Alexander (1996) apresentam é que apesar de todas as vantagens do usos de padrões, há coisas que eles não podem fazer. Como padrões surgem de experiências, quando novas tecnologias surgem pode acontecer de não existirem padrões que solucionem os problemas dessa nova tecnologia. As soluções para esses problemas irão surgir com o tempo e com isso surgirão também padrões. Padrões são feitos para servir de guias para humanos e não para máquinas. Nesse sentido, eles não vão gerar códigos e nem resolver problemas sozinhos. Dessa forma, eles não devem substituir programadores capacitados. Apesar de padrões serem frutos de conhecimentos de especialistas, eles não transformam novatos em especialistas. Os padrões ajudam no dia-a-dia dos desenvolvedores para que eles não cometam erros simples e os convidam a tentarem novos princípios.

Pressman (2011) diz que a maioria dos problemas tem várias soluções. Cabe ao projetista analisar o projeto em questão e verificar o que mais atente a sua necessidade. Com isso, um padrão eficaz tem as seguintes características:

1. **Um padrão soluciona problemas:** Padrões descrevem uma solução, com isso ele não deve descrever estratégias ou princípios abstratos;

2. **Um padrão é um conceito comprovado:** Eles contém soluções que já foram aplicadas várias vezes e não soluções com que são teóricas ou especulações;

3. **Uma solução não é óbvia :** Um bom padrão gera uma solução para um problema indiretamente. Isso é necessário para problemas mais difíceis, pois as soluções desses problemas não são óbvias;

4. **Um padrão descreve uma relação :** Os padrões não apenas descrevem módulos, como também estruturas e mecanismos de sistemas mais profundos.

5. **Um padrão possui um componente humano significativo :** Todo software visa aumentar o conforto e a qualidade de vida. Nesse sentido, bons padrões seguem essa lógica apelando para a estética e usabilidade.

### 2.4.1 Tipos de Padrões

De acordo com Pressman (2011), engenheiros têm um grande interesse em padrões, pois seres humanos são muito bons em reconhecer padrões. Com isso, é natural que os desenvolvedores tenham reconhecido padrões de comportamento entre os padrões e os tenham separado em tipos:

- **Padrões de arquitetura:** Eles descrevem problemas de projeto de caráter diverso e amplo, resolvidos utilizando uma abordagem estrutural;
- **Padrões de dados:** Descrevem problemas orientados a dados recorrentes e as soluções de modelagem de dados que podem ser usadas para resolvê-los;
- **Padrões de Componentes ou Padrões de Projeto:** Tratam de problemas associados ao desenvolvimento de subsistemas e componentes. Como eles comunicam entre si e seu posicionamento em uma arquitetura maior;
- **Padrões de Interface de usuário:** são recorrentes soluções que resolvem problemas de *design* comuns. Eles são referências para *designers* de interfaces. Eles fornecem uma linguagem comum entre *designers*;
- **Padrões para *WebApps*:** Tratam de um conjunto de problemas encontrados para se construir *WebApps* e , em geral, englobam os padrões mencionados;
- **Padrões Criacionais:** Concentram - se na criação, composição e representação de objetos. Eles dispõe de mecanismos que facilitam a instanciação de objetos em um sistema e impõe restrições sobre o tipo e o número de objetos que podem ser criados;
- **Padrões Estruturais:** Focalizam problema e soluções relacionados a como classe e objetos são organizados e integrados para formar uma estrutura maior. Em essência, ajudam a estabelecer relações entre entidades em um sistema;
- **Padrões Comportamentais:** Tratam de problemas associados a atribuição de responsabilidades entre objetos e a maneira pela qual a comunicação é efetuada entre esses objetos.

### 2.4.2 Tipos de Padrões de Interface

Com o passar do tempo as necessidades dos usuários foram mudando. Com isso, surgiram vários subtipos de padrões a partir dos tipos mostrados. Esse trabalho, irá se focar nos padrões de interface. De acordo com Neil (2014), alguns dos tipos de padrões de interface são:

- **Navegação:** Uma boa navegação é como um bom *design*: é invisível ao usuário. Aplicações que têm essa característica são intuitivas e disponibilizam uma forma simples de realizar qualquer ação. Desde fazer um cadastro a gerenciar uma lista de contatos;
- **Formulários:** Os formulários são a forma mais comum de adquirir dados de usuários em aplicações *web*. Com isso, muitos padrões para formulários surgiram, como por exemplo, os padrões de interface para o login do usuário em algum sistema;

- **Tabelas:** Muitas aplicações empresariais exigem a exibição de uma grande quantidade de dados. Com isso, ao longo do tempo os desenvolvedores criaram muitas soluções para exibir essa grande quantidade de dados em tabelas, de forma clara e elegante;

- **Pesquisar, Classificar e Filtrar:** Realizar essas funções parece ser simples, no entanto realizá-las com eficiência e de forma com que o usuário fique satisfeito exige um *design* bem pensado;

- **Ferramentas:** Com o aumento da complexidade das aplicações, foi necessário que as ferramentas utilizadas por elas se tornassem mais simples e intuitivas. Com isso, muitas ideias surgiram através dessa necessidade e as ideias que se solidificaram, tornaram-se padrões. Um exemplo de padrão é o *toolbar*, que mostra ao usuário uma lista de ferramentas que ele pode usar para manipular a aplicação. Muitas ferramentas de edição de imagens utilizam esse padrão;

- **Tutorias e Convites:** Tutorias são extremamente necessários atualmente, pois as aplicações estão cada vez mais complexas e o usuário não pode ser obrigado a entender as funcionalidades de uma aplicação sozinho. Dessa forma, padrões para tutoriais surgiram com o intuito de ajudar nesse problema. Os convites são outra forma de os usuários se familiarizarem melhor com uma aplicação, pois eles convidam o usuário a utilizar funcionalidades que ainda não tem conhecimento. Com isso, vários padrões surgiram para eles também;

Além disso, no anexo A existe uma forma com que uma estrutura de padrões pode ser organizada.

# 3 TRABALHOS RELACIONADOS

Para se conhecer o estado atual da arte, foi necessário se buscar trabalhos relacionados aos assuntos abordados nesse artigo. A busca de artigos foi feita através das bibliotecas digitais: IEEE, Scopus, ACM e Science Direct. Como foram encontrados poucos trabalhos relacionados aos Padrões de interface, uma busca mais abrangente foi necessária. Dessa forma, para complementar a busca, foram pesquisados artigos utilizando o Google Scholar.

Essa Seção descreve esses trabalhos e demonstra quais as diferenças entre eles e o nosso trabalho. Primeiros são demonstrados trabalhos relacionados ao aprendizados de jogos. Posteriormente são apresentados trabalhos que têm relação com o *design* de jogos educativos. Em seguida são apresentados os trabalhos relacionados a padrões de interface.

## 3.1 Aprendizado Através de Jogos

No artigo de Hernandez et al. (2010) os autores propõe o ensino de programação através de uma *engine* de jogos. De acordo com eles, os alunos tinham muita dificuldade em aprender a programar no curso de Ciência da Computação. Nesse sentido, os autores sugeriram realizar o ensino de programação aos alunos através da criação de jogos, pois os jogos fazem parte do cotidiano desses alunos. Dessa forma, realizaram um estudo de caso com os alunos da universidade do Cruzeiro do Sul para verificar se a técnica proposta funcionaria. Para realizar esse estudo, eles utilizaram uma *engine* de jogos chamada GameMaker, pois essa *engine* disponibilizaria uma forma fácil dos alunos visualizarem os resultados de sua implementação. Os autores chegaram a conclusão que o uso dessa técnica aumentou a performance dos alunos, comparado ao ano em que eles não aplicaram a técnica em sala de aula. Esse trabalho mostra que o uso de conceitos familiares para os alunos ajuda no aprendizado deles.

Já no artigo de Chau et al. (2014), os autores descrevem o uso de uma API, chamada Corrupted, para o ensino de conceitos de programação relacionado a vetores. Essa API dá aos alunos a capacidade de criar jogos interessantes e visualmente ricos, utilizando poucas linhas de código. Um exercício que eles utilizam é o *Line clear*. Nele, os estudantes utilizam a API para popular um vetor. A cada atualização do jogo, os estudantes procuram eventos relacionados ao click com botão direito ou esquerdo do mouse. Dependendo do botão clikado e da posição do mouse, a API aciona um método implementado pelos alunos para lidar com o evento. Esse exercício, possibilita que os alunos pratiquem o percorrimento de vetores, enquanto a API demonstra animações do resultado do código implementado pelos alunos. Os autores acreditam que com o uso dessa API, os alunos irão se manter mais motivados. No entanto, os autores não apresentaram evidências de que essa API realmente funcione.

O artigo de Eagle e Barnes (2008) descreve um jogo chamado Wu's Castle, desenvolvido para que alunos aprendam sobre vetores e laços. O jogo foi desenvolvido na

linguagem Ruby, utilizando o software RPG Maker XP. No jogo, o jogador tem diversos objetivos. Para completar esses objetivos, o jogador deve interagir com vetores e laços, fazendo escolhas e definindo variáveis. Para verificar a efetividade do jogo no aprendizado dos alunos, os autores fizeram testes antes e depois deles jogarem. Os autores verificaram que os alunos que jogaram, tiveram um aumento de aprendizado significativo comparado aos alunos que não jogaram. Esse estudo mostra que um jogo projetado para o ensino de programação realmente pode ajudar alunos a se manterem motivados e a aprenderem programação mais rápido.

No artigo de Rankin, Gooch e Gooch (2008) é analisado o impacto de jogos no interesse de estudantes da Ciência da Computação. Para isso, os autores selecionaram 56 estudantes de graduação do curso de Ciência da Computação para projetar e implementar um jogo 2D utilizando a plataforma Game Maker. Para verificar os resultados, os autores realizaram pesquisas antes e depois da criação do jogo. Os autores verificaram que utilizar a criação de jogos para ensinar programação aumentou o interesse do alunos, além de fazer com que eles aprendessem técnicas de Engenharia de Software, como controle de versão. Além disso, os alunos também adquiriram habilidades em outras áreas, como Músicas, Arte e Animações. No entanto, eles identificaram que os alunos precisam de tempo para que o jogo resultante seja satisfatório, pois o projeto foi realizado só em duas semanas. Portanto, os autores identificaram que no geral, os alunos se esforçaram mais e ultrapassaram os objetivos de aprendizado propostos, no contexto de desenvolvimento de jogos.

Em contrapartida dos artigos mostrados até agora, o artigo de Adams, MacNamara e Richard Wainess (2011) encontrou que os alunos aprenderam mais a partir de *slides* com o conteúdo do que com um jogo educativo. O objetivo do artigo foi testar se a narrativa e o descobrimento de conteúdos em jogos educativos ajudam no aprendizado. Os autores realizaram dois experimentos. No primeiro experimento é utilizado um jogo chamado Crystal Island, em que o jogador tem descobrir a causa de uma doença que aconteceu numa ilha remota. No segundo experimento, o jogo utilizado foi o Cache 17, em que os jogadores têm que descobrir o destino de um esconderijo cheio de pinturas que desapareceram durante a segunda guerra mundial. Para se obter os resultados, os alunos que aprenderam o conteúdo através do jogo eram comparados com alunos que aprenderam o conteúdo através de slides. Os autores encontraram que os alunos que jogaram os dois jogos não tiveram uma grande diferença de aprendizado. No entanto, os alunos que usaram os slides para aprender, tiveram mais sucesso do que os alunos que jogaram o jogo educativo. Esse artigo mostra que nem sempre os jogos educativos são eficientes para o ensino. No entanto, o conteúdo que os autores ensinavam, tem uma base mais teórica, comparado com o conteúdo mais prático mostrado nos outros artigos com resultados positivos. Com isso, é possível que o tipo de conteúdo ensinado afete no aprendizado do aluno, quando se é utilizado jogos educativos.

Nos artigos apresentados até agora, os autores buscam aumentar o interesse do aluno em programação e ensinar o aluno princípios básicos de programação. O jogo proposto por esse trabalho tem como objetivo demonstrar ao aluno conceitos de Engenharia de Software e ser um ambiente de prática de programação, além dos objetivos apresentados.

## 3.2 Design de Jogos

O artigo de Dickey (2005) compara e analisa como as estratégias utilizadas nos jogos para manter o jogador interessado podem ser utilizados por instrutores para manter seus alunos interessados no conteúdo. Ele detalha como técnicas como o posicionamento do jogador, a narrativa e o design interativo podem ajudar o instrutor a manter o interesse dos alunos. O autor não chega a uma conclusão definitiva das estrategias que funcionariam para instrutores. No entanto, o artigo sugere que é possível se juntar aprendizado com diversão em um jogo educativo.

## 3.3 Padrões de Interface para Jogos

No artigo de Folmer (2006), o autor cria um catálogo de padrões de usabilidade para jogos. Ele os identificaram utilizando jogos existentes e discutindo com desenvolvedores. Um exemplo de padrões estruturados por ele é:

- **Nome do Padrão:** Nível de dificuldade adaptado.
- **O problema:** O jogo é muito difícil de se jogar na configuração atual. O jogador perde várias vezes seguidas
- **Contexto:** Jogos que oferecem diferentes níveis de dificuldade para atender a diferentes tipos de jogadores. A maioria dos termos comuns para níveis de dificuldade são fáceis, normal e difícil. Dependendo o nível de dificuldade:
  - Inimigos são mais fortes ou mais fracos;
  - Enigmas são mais fáceis ou mais difíceis de resolver;
  - Mais ou menos orientação é disponibilizada.
  - Mais ou menos assistências nos controles são disponibilizadas.

  Normalmente, o jogador escolhe um nível de dificuldade ao iniciar o jogo, e geralmente é não é possível mudar para um diferente durante o jogo.
- **Forças:**

  - Os jogadores querem o jogo que ofereça um desafio. No entanto, eles ficam frustrados quando chegam em uma parte em que têm que ficar tentando várias vezes.
  - Alguns jogadores podem ter experiência jogando outros jogos parecidos, outros podem ter pouca.
  - Alguns jogadores são piores ou melhores que outros em certos aspectos de um jogo, como por exemplo atirar ou resolver enigmas.
  - A dificuldade de um aspecto do jogo pode variar durante o jogo, como por exemplo uma parte do jogo pode focar mais em atirar e outra pode focar mais em resolver enigmas.

- **A solução:** Ajustar o nível de dificuldade ao jogador. O jogo deve se adaptar a diferentes jogadores durante diferentes partes do jogo. Existem duas opções de execução do presente padrão. Sugerir um nível de dificuldade diferente de acordo com o desempenho do jogador. Por exemplo. o jogo poderia sugerir uma dificuldade mais fácil quando o jogador falhar em passar de um ponto várias vezes seguidas. A outra opção ajustar automaticamente o nível de dificuldade com base no desempenho do jogador. Isso remove os "níveis de dificuldade tradicionais". Cada vez que o jogador morre ou falhar, existe uma chance de que o jogador vai mudar para um nível mais fácil. Se jogar bem por um tempo, há uma chance de o jogador vai subir um nível de dificuldade.

- **Princípio:** Flexibilidade e eficiência de uso.

- **Por quê:** Ajustar o nível de dificuldade para o jogador pode aumentar a satisfação e eficiência e os jogadores não ficarão frustrados.

- **Exemplos:** *God of War*, *Resident Evil 4*.

A diferença entre o artigo discutido e este trabalho é que Folmer (2006) foca na usabilidade de jogos que, de acordo com ele, está mais relacionado a jogabilidade de um jogo do que com a interface. Já este artigo irá focar nos padrões de interface, ou seja, em como as informações são apresentadas ao jogador e como ele manipula essas informações.

Já no artigo de Fagerholt e Lorentzon (2009), o autor propões *guidelines* para a criação de uma interface em jogos *First Person Shotter (FPSs)*. Ele analisou vários jogos existentes e a partir disso gerou 4 *guidelines* para orientar desenvolvedores. A primeira é : conheça o espaço do seu jogo. Ela diz que o desenvolvedor deve estudar o espaço do seu jogo e como uma interface pode ser desenvolvida nesse espaço. Muitos jogos são em 3D. Dependendo do jogo, a interface pode ser projetada para ser 3D também. Um exemplo disso são os jogos futuristas, em que a interface podem ser representações de objetos tecnológicos. A outra *guideline* é: conheça o seu jogo. Ela diz que o desenvolvedor tem

que estar atento ao que o jogo dele propõe. Se o jogo for mais focado na competição entre jogadores, talvez existam informações extras que o jogador deva saber e para exibi-las o desenvolvedor tem que retirar alguns aspectos de imersão. Essa *guideline* também diz que conhecer o contexto do jogo é importante, pois através desse dele é possível criar interfaces que aumentem a imersão do jogar ao invés de tira-la, como no exemplo já citado de um jogo futurista. A terceira *guideline* é: estabelecer controle do jogador. Com o aumento da tecnologia, os jogos atualmente têm uma grande fidelidade ao mundo real. Com isso, os jogadores começam a jogar um jogo com grandes expectativas sobre o que ele pode interagir dentro dele. Isso se torna um problema para o desenvolvedor, pois o jogador pode tentar interagir com coisas que não foram projetadas para interação ou podem jogar de forma errada o jogo. Com isso, cabe aos desenvolvedores estudar formas de guiar o jogador sobre o que ele pode ou não interagir. No entanto, deve-se tomar cuidado para não limitar demais as ações do jogador e tirar a sensação de liberdade dele. A última *guideline* é: fortalecer o link entre jogador e *avatar*. Ela diz que muitos dos sentidos humanos não podem ser transportados para o mundo virtual com a tecnologia atual. Uma sugestão do autor é utilizar a interface para ilustrar sensações do avatar do jogador, como por exemplo deixar a tela vermelha se o avatar está com dor. Outro problema relacionado a perda dos sentidos é que o jogador pode perder o senso de direção quando se está jogando. Para resolver isso, o autor sugere colocar pistas sobre a navegação do jogador, como por exemplo colocar pegadas onde o ele já passou.

O artigo do Fagerholt e Lorentzon (2009) estabelece bons guias para se fazer uma interface em jogos FPS. No entanto, ele é muito focado em um gênero. Uma diferença dele para este trabalho é que focaremos em um contexto mais geral, sem especificar um gênero para os padrões estabelecidos. Outra diferença é que no artigo, são apresentadas *guidelines* para o desenvolvedor pensar sobre em como desenvolver sua interface, sem apresentar uma solução. Neste trabalho, a intenção é apresentar padrões, com soluções que o desenvolvedor possa aplicar em seu projeto.

Infelizmente, não foram encontrados muitos artigos que tratam do assunto de padrões de interface sendo aplicados nos jogos. Mas observando os artigos encontrados, nota-se que houveram tentativas de propor *guidelines* e Padrões de Usabilidade para serem aplicados em jogos. Nesse sentido, esses trabalhos poderão ajudar na criação de um catálogo de Padrões de Interface. Além disso, os artigos relacionados a criação de jogos educativos mostram várias formas de se criar um jogo com foco na educação e isso pôde servir de inspiração para a criação do jogo proposto por esse trabalho.

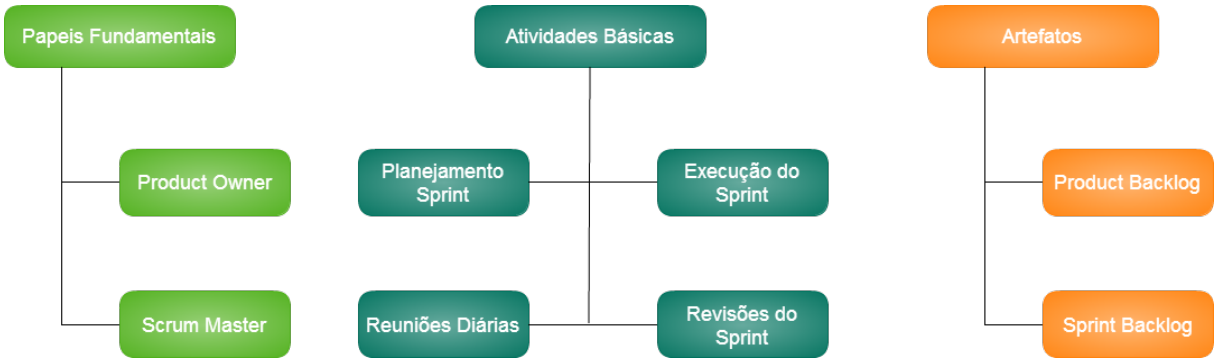
# 4 METODOLOGIA

Nesta Seção é apresentada a forma com este trabalho foi executado. Primeiro é detalhado como o processo de desenvolvimento aconteceu. Depois, é detalhado como a coleta de padrões ocorreu e por último há a detalhamento de como o processo de avaliação do jogo ocorreu.

## 4.1 Processo de Desenvolvimento

O jogo proposto foi desenvolvido pelo autor desse trabalho com a ajuda do seu orientador.Para se desenvolver o jogo é necessário o uso de um processo de desenvolvimento, pois o jogo proposto apresenta várias funcionalidades a serem implementas e um processo manteria o desenvolvimento organizado. Com isso, o processo escolhido foi uma variação do método Scrum, pois ele apresenta uma forma ágil para se gerenciar tarefas que se encaixa com a realidade do desenvolvedor do jogo. As alterações no processo foram necessárias, tendo em vista que só há um desenvolvedor envolvido e não uma equipe de desenvolvimento.

Figura 1 – Práticas do Scrum no formato original




Fonte: Própria

Na Figura 1 estão ilustradas práticas adotadas no Scrum em que, de acordo com Schwaber (1997), os papeis são o dono do produto, *Scrum Master* e os desenvolvedores. O dono do produto é responsável por cuidar do artefato *Product Backlog*. O *Product Backlog* é uma lista de requisitos que devem ser implementados para o sistema ficar pronto. O dono do produto prioriza os requisitos presentes nele e valida as entregas dos desenvolvedores. O *Scrum Master* é responsável por garantir que todas as atividades básicas do Scrum funcionem. A primeira atividade a ser realizada é o planejamento do *Sprint*, em que os desenvolvedores irão escolher quais requisitos irão implementar, com base nas prioridades definidas pelo Dono do produto. A partir dessa atividade surge o *Sprint Backlog*, que contém todas as funcionalidades a serem desenvolvidas naquele *Sprint*. Após o planejamento, começa a execução do *Sprint* que é a implementação dos requisitos definidos no *Sprint Backlog*. Durante a execução, acontecem reuniões diárias para acompanhar o

desenvolvimento. Quando o prazo do *Sprint* acabar, os desenvolvedores deverão entregar o produto para o dono do produto validar. Antes de um novo *Sprint* começar, deve-se ser feita a revisão do Sprint, em que o grupo de desenvolvedores irá discutir formas de melhorar o desenvolvimento para o próximo *Sprint*. Como o Scrum é processo iterativo e incremental, essas atividades são realizadas várias vezes até que o produto esteja pronto.

No processo adaptado, o orientador agiu como o dono do produto. O papel dele foi de avaliar as funcionalidades e definir prioridades para elas. Com isso, as funcionalidades de prioridade maior forão implementadas primeiro. O aluno assumiu o papel de desenvolvedor, sendo responsabilidade dele cumprir todas as tarefas definidas pelo orientador. Além disso, o desenvolvedor foi responsável por popular o *Product Backlog* com requisitos e outras tarefas. Esses requisitos foram escritos no formato de estórias de usuário, em que se deve especificar o ator, a ação e a funcionalidade desejada, por exemplo: como jogador quero selecionar um desafio a ser feito. O papel de *Scrum Master* não existe nesse Scrum adaptado, pois na equipe só existe um desenvolvedor. As entregas das funcionalidades foram semanais, ou seja, a cada semana o desenvolvedor entregou novas funcionalidades e o dono do produto validou as entregas e revisou as prioridades. A ferramenta utilizada para organizar esse processo foi o Trello[1].

Para desenvolver o jogo uma *engine* chamada *Cocos Creator* foi utilizada, ela é uma *engine* de jogos que poder gerar jogos para diferentes plataformas. Nesse caso, só utilizamos a plataforma web. Além disso, para o *Heroku* foi utilizado para implantar o jogo. O *Heroku* é um serviço que possibilita os desenvolvedores criar, executar e operar aplicações inteiramente na nuvem. Nesse sentido, o *Heroku* pode ser usado de graça, de forma limitada.

## 4.2 Coleta de Padrões

Um dos objetivos desse trabalho é identificar e estruturar padrões de interface voltados a jogos e aplicá-los ao jogo desenvolvido neste trabalho. Dessa forma, a coleta desses padrões foi realizada através de buscas em repositórios, verificando os padrões de interface já estabelecidos. Nesse sentido, não será proposta a criação de padrões. A busca por esses repositórios aconteceu primeiramente em periódicos (Capes, IEEE, GoogleScholar), utilizando as palavras chaves: games user interface patterns, Games to teach programming, educational games. Foram encontrados padrões de usabilidade e diretrizes de interface para jogos (FAGERHOLT; LORENTZON, 2009; FOLMER, 2006). Posteriormente, foi realizada uma pesquisa mais ampla no motor de buscas Google pelas mesmas palavras-chaves utilizadas anteriormente. Essa pesquisa levou a repositórios não estruturados de padrões de interface para jogos (ADAM, 2013; ALEXANDER, 2009; TANG, 2015) que foram usados também nesse trabalho.". Para documentar os padrões, é utilizada uma

[1] Link do Trello: <https://trello.com>

versão adaptada da estrutura proposta por Gamma (2007), como explicado na Seção A. Foi necessário retirar os tópicos que não se aplicam a interface, como os tópicos: estrutura, participante e colaboração. A estrutura escolhida é apresentada abaixo.

- **Imagens:** É demonstrada uma imagens do padrão aplicado.
- **Nome do Padrão:** Aqui é descrito o nome do padrão;
- **Intenção:** Aqui é especificado qual a é a intenção do padrão para se usar esse padrão;
- **Motivo:** Aqui se explica a motivação para se usar o padrão;
- **Aplicabilidade:** Aqui se explica quando se utilizar o padrão;
- **Consequências:** Aqui se mostra quais são as consequências de se aplicar o padrão, incluindo benefícios e malefícios;
- **Implementação:** Aqui se mostra como é feita a implementação desse padrão;
- **Usos Conhecidos:** Aqui se mostra quais os produtos que já utilizam esse padrão;

## 4.3 Processo de Validação do Jogo

Quando há a criação de um projeto, é importante que haja algum processo de validação para verificar se ele realiza as funções que se propõe. Além disso, uma validação pode resultar em *feedbacks* positivos para o futuro do projeto. Nesse sentido, resolvemos utilizar alunos e professores da Unipampa para realizar uma validação do jogo proposto. Com isso, foi possível validar a eficácia do jogo como ferramenta de educação, analisar como a aplicação de padrões afeta na experiência que os alunos têm ao jogar e receber *feedbacks* do público alvo. Para isso, resolvemos utilizar o questionário MEEGA+ proposto por Ripasy et al. (2018), que serve para avaliar jogos educacionais na perspectiva dos alunos e professores. No entanto, algumas alterações foram feitas nas partes dos professores, pois o MEEGA+ propõe um questionário do professor que avalia tanto sua a perspectiva quanto ao jogo, quanto a impressão que ele teve ao aplicar o jogo para os alunos. Nesse sentido, a segunda parte do questionário relacionado a aplicação não se encaixa no nosso contexto, pois a validação seria aplicada somente pelo pesquisador e queríamos ter o *feedback* de mais professores. Com isso, a segunda parte do questionário foi retirada para os professores que não iriam aplicar o jogo diretamente para os alunos.

A aplicação da validação do jogo com os alunos foi dividida em duas partes, em que na primeira eles experimentaram o jogo e na segunda eles responderam ao questionário do MEEGA+, sendo que a experiência com o jogo foi de 30 minutos. Um total de 48 alunos participaram e a avaliação foi feita no laboratório 4 da Unipampa. Esse questionário

estava em um formulário do *Google Docs*[2]. Já na aplicação com os professores, eles jogaram o jogo e o pesquisador acompanhou o processo. Eles preencheram o questionário, que também utilizava o *Google Docs*, e o pesquisador fez anotações de opiniões expressadas por eles. Um total de 5 professores participaram da avaliação e ela foi feita na salas de cada respectivo professor.

[2] Link do formulário: <https://docs.google.com/forms/d/e/1FAIpQLScI-2abCFIGKq3xQyqv3bt3fuJ4TSB_WD71IlZmckuVf_Or_w/viewform>

# 5 RESULTADOS

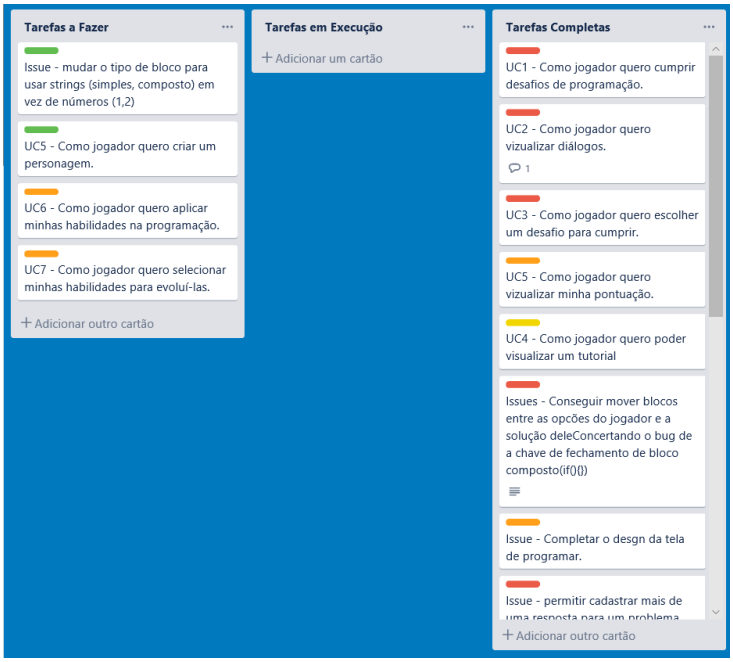
Nesta seção, será apresentado os resultados obtidos durante a realização do TCC. Primeiro iremos falar sobre a implementação do jogo Programmer, detalhando suas mecânicas, decisões de *design* e suas principais funcionalidades. Depois, iremos falar sobre a catalogação e o uso dos padrões de interfaces coletados durante a execução do TCC 1, detalhando as suas funções e a aplicação deles dentro do jogo. Por último, iremos falar sobre os resultados da validação aplicada.

## 5.1 Programmer

### 5.1.1 Execução do Scrum

Como apontado na metodologia, o Programmer foi desenvolvido utilizando o Scrum. Nesse sentido, para organizar os processos do Scrum, o Trello foi utilizado. As tarefas a serem realizadas durante o *Sprint* eram classificadas em prioridades, sendo elas: mínima(cor verde), baixa(cor amarela), média(cor laranja) e alta(cor vermelha). Essas prioridades, eram estabelecidas pelo PO, que é responsável por gerenciar os requisitos do jogo. As tarefas eram organizadas em três colunas, sendo que a primeira representava as tarefas a ser feitas, a segunda as tarefas em execução e a terceira as tarefas concluídas. Além disso, problemas que apareciam durante o desenvolvimento eram documentados no Trello como *issues*. A figura 2 mostra os resultados da aplicação do Scrum durante o desenvolvimento do jogo. Essa aplicação aconteceu ao realizar reuniões semanais, em que o orientador analisava o resultado do *Sprint* atual e um novo era planejado a partir dai.

Figura 2 – Atividades realizadas organizadas no Trello



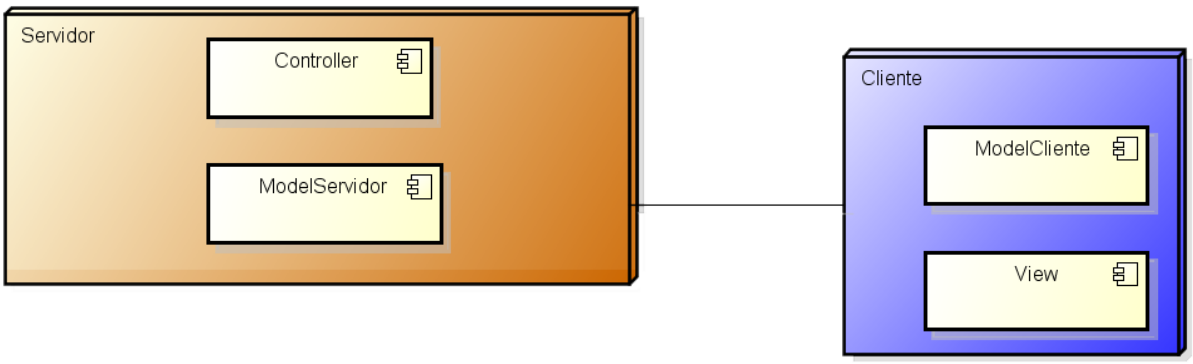

Fonte: Própria

### 5.1.2 Visão Geral

A plataforma escolhida para o jogo foi *web*, pois faria com que o ele conseguisse atingir mais usuários, por rodar em navegadores. Com isso, foi necessário se pensar em uma arquitetura para o jogo. A Figura 3 apresenta um diagrama de implantação utilizado para demonstrar a arquitetura. O nó representa um recurso computacional do sistema, nesse caso são o servidor e os clientes. Os componentes representam as camadas dele.

- O servidor tem:
  - A camada *Controller*, responsável por controlar as operações do sistema;
  - A camada *Model*, responsável pela parte lógica do sistema no servidor;
- O Cliente tem:
  - A camada *Model*, responsável pela parte lógica do sistema no cliente;
  - A camada *View*, que apresenta o sistema ao usuário de forma visual.

Figura 3 – Diagrama de Implantação demonstrando a arquitetura




Fonte: Própria

Devido ao sistema funcionar em mais de um ambiente, a camada *Model*, precisa de uma representação no servidor e no cliente. Isso mantém a consistência dos dados durante o uso do jogo.

O jogo é dividido em desafios, sendo que um desafio representa um algoritmo que o usuário tem de montar para resolver um problema proposto. Com isso, o servidor é responsável por disponibilizar os desafios e verificar se o usuário os terminou. Já o cliente tem todas as outras mecânicas do jogo. A escolha da criação do servidor foi feita para evitar que o usuário pudesse trapacear, olhando as respostas no HTML. Dessa forma, a representação do desafio no cliente e no servidor é feita de forma diferente. A diferença é que no servidor o desafio contém as possíveis respostas do usuário. A figura 4 mostra parte de como um desafio é representado no servidor, através de um arquivo JSon. O desafio é representado da seguinte forma:

- **Nome:** Nome do desafio;
- **Descrição:** Descreve qual o problema que o usuário tem de resolver;
- **Tempo Inicial:** Tempo que o usuário começa a resolver o desafio;
- **Tempo Final:** Limite de tempo que o usuário irá ter para resolver o desafio;
- **Dificuldade:** dificuldade do desafio;
- **Opções:** Opções que o usuário tem para resolver o desafio;
- **Solução:** Solução que o usuário monta para o problema;
- **Respostas(Somente Servidor):** Contém as possíveis respostas para o desafio.

Figura 4 – Desafio representado no servidor através de um JSON


```
{
  "id": "2",
  "nome": "Média Aritmética",
  "descricao": "Criar um algoritmo que imprima a média aritmética entre os números 8, 9 e 7.",
  "opcoes": [
    {
      "id": "1",
      "valor": "media = (8 + 9 + 7) / 3;",
      "tipo": "1",
      "blocosFilhos": []
    },
    {
      "id": "2",
      "valor": "float media;",
      "tipo": "1",
      "blocosFilhos": []
    },
    {
      "id": "3",
      "valor": "printf(\"media = %f\" , media);",
      "tipo": "1",
      "blocosFilhos": []
    }
  ],
  "solucao": [
    {
      "id": "100",
      "valor": "void main()",
      "tipo": "2",
      "blocosFilhos": []
    }
  ],
  "respostas": [
    {
      "blocosResposta": [
        {
          "id": "100",
          "valor": "void main()",
          "tipo": "2",
          "blocosFilhos": [
            {
              "id": "2",
              "valor": "float media;",
              "tipo": "1",
              "blocosFilhos": []
            },
            {
              "id": "1",
              "valor": "media = (8 + 9 + 7) / 3;"
```

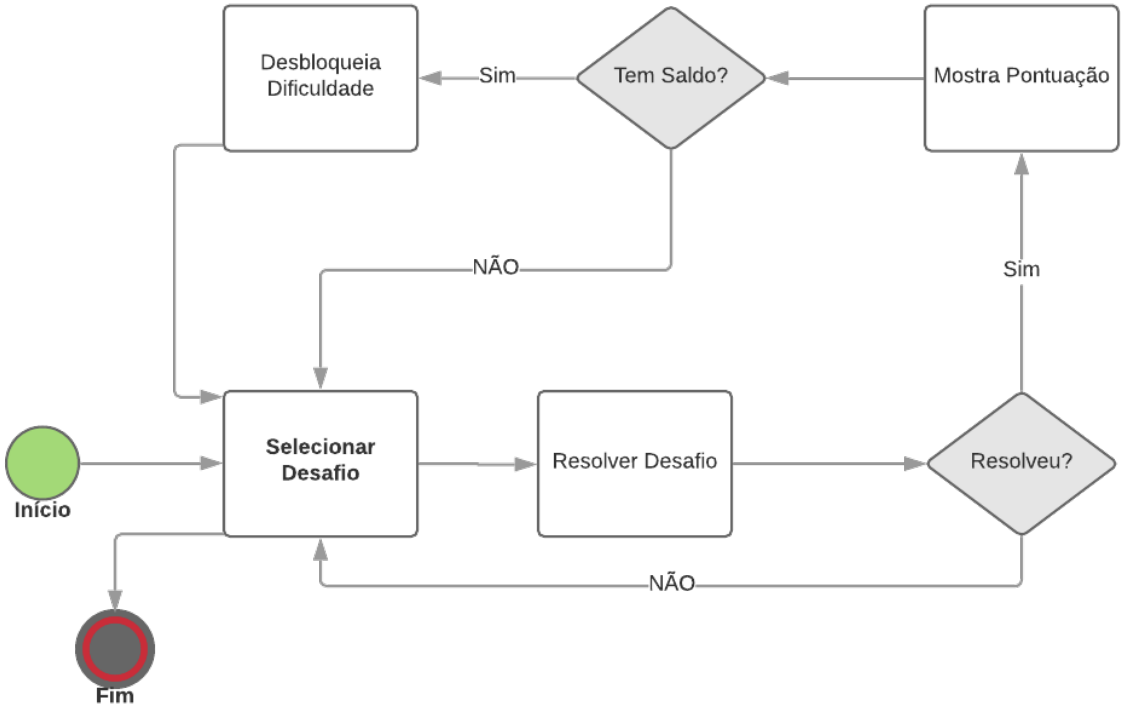

Fonte: Própria

As opções e solução do jogador são compostas de blocos. Um bloco é uma representação de um comando de código. Esses blocos podem ser de dois tipos, sendo o primeiro simples e o segundo composto. Um bloco simples configura um código que não se estende por mais de uma linha, como por exemplo uma declaração de variável “int x

= 10;". Já um bloco composto, representa códigos que se estendem a mais de uma linha, além de poderem ter outros códigos dentro dos mesmos. Um exemplo de código composto é "if(x == 10)", pois um *if* pode ter linhas de códigos dentro dele e pode ocupar mais de uma linha.

O objetivo do jogador no desafio é montar um código que resolva um problema proposto. Para isso, o jogador deverá escolher entre os blocos de códigos e montá-los na ordem certa. No entanto, o jogador deverá completar o código dentro de um tempo determinado. Quanto mais rápido o jogador completar o código, mais o jogador será recompensado. O jogador receberá *feedback* de suas ações, além de receber dinheiro quando completar os desafios. Esse dinheiro poderá ser usado para desbloquear novas dificuldades. O dinheiro é informado através de uma tela de pontuação que, além disso, dá uma nota a performance do jogador. Essas funcionalidades juntas formam o *loop* do jogo, em o jogador resolve desafios, ganha dinheiro como recompença e desbloqueia novos desafios. A figura 5 apresenta *loop*:

Figura 5 – Loop do jogo

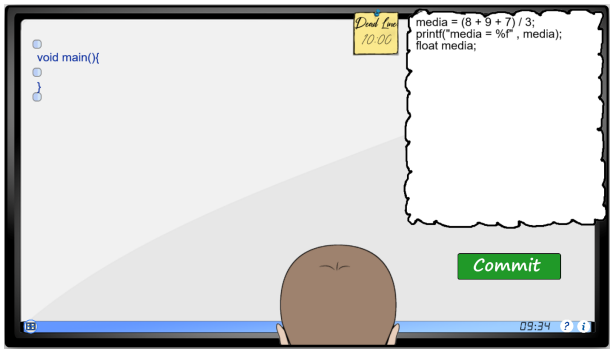



Fonte: Própria

## 5.2 Padrões Aplicados

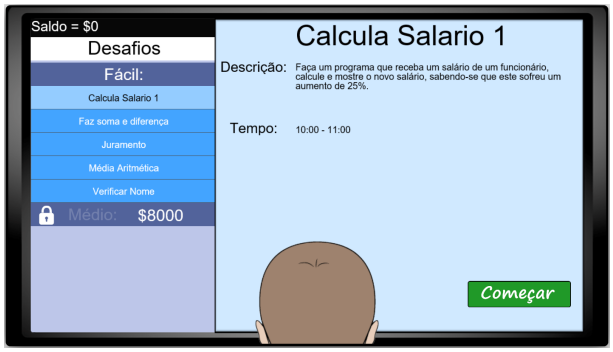
### 5.2.1 Skeuomorphic

Figura 6 – Padrão Skeuomorphic sendo aplicado na tela de Programação



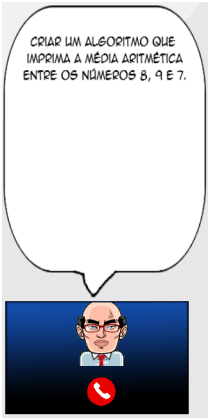

Fonte: Própria

Figura 7 – Padrão Skeuomorphic sendo aplicado na tela de Seleção de desafios




Fonte: Própria

Figura 8 – Padrão Skeuomorphic sendo aplicado na comunicação do personagem “chefe” com o jogador.




Fonte: Própria

- **Nome do Padrão:** Skeuomorphic;
- **Intenção:** Mostrar interfaces ao jogador sem que elas atrapalhem na sua suspensão de descrença;
- **Motivação:** Em jogos mais complexos, é necessário se criar interfaces extras para auxiliar o jogador ou para que ele possa realizar tarefas. Um dos grandes problemas quando se está projetando esse tipo de interface para um jogo é a necessidade de não afetar a suspensão de descrença do jogador ao acessar essas interfaces;
- **Aplicabilidade:** Esse padrão deve ser usado quando o desenvolvedor precisar mostrar a um jogador uma interface muitas vezes durante o jogo e não quiser que isso interfira em sua suspensão de descrença;
- **Consequências:**
  - O design da interface pode ficar limitado por causa da intenção de manter a fidelidade ao mundo real;
  - Aumenta a suspensão de descrença do jogador.
- **Implementação:** Fazer o *design* da interface espelhando um objeto ou experiência da realidade do qual o item é inspirado. Ao implementar esse padrão é necessário se olhar para o contexto do problema e analisar como é possível se usar elementos da realidade para fazer a solução ficar mais intuitiva para o jogador. Na Figura 6 é demonstrado um exemplo de uso desse padrão, em que era necessário desenvolver uma interface para que o jogador pudesse resolver os problemas de programação propostos pelo jogo. Dessa forma, a interface foi pensada usando as experiências de um programador real.

  A interface ficou dividida em duas partes: o que acontece no computador e o que acontece dentro da mente do jogador. O computador é representado pelo monitor, dentro dele o jogador pode colocar blocos de código para desenvolver uma solução para o problema. Além disso, temos uma barra na parte de baixo do monitor que representa uma barra de tarefas do sistema operacional Windows. Nela, temos, da esquerda para a direita, o botão de opções, o horário atual do jogo, o botão de ajuda e o botão para ver informações sobre o desafio. O último elemento do monitor, é o *post it* posicionado na parte de cima da tela. Ele foi utilizado, pois é uma ferramenta muito usada em ambientes de desenvolvimento de softwares. Nele, é demonstrado o *deadline* do jogador, ou seja, o limite de tempo que ele terá para resolver o desafio. Apesar do tempo passar mais rápido no relógio que está nessa tela, ele foi representado em formato de hora para manter um visual parecido com a barra de tarefas do Windows.

Já o programador é representado pela cabeça na parte de baixo da imagem, em que ele tem um balão de pensamento que representa suas ideias e mostra as opções disponíveis para que ele resolva o problema. Juntando essas duas partes, a criação de uma solução dentro do jogo fica parecida com a experiência de um programador de verdade.

Na figura 7, temos um exemplo do padrão skeuomorphic sendo aplicado na seleção de desafios, em que essa tela foi pensada no intuito de parecer uma caixa de e-mails. Cada desafio representa um e-mail do chefe com um problema a ser resolvido. Outro exemplo está na figura 8, em que toda a comunicação do jogador com o personagem "chefe" é feita por uma representação de uma chamada de Skype. Fizemos essa decisão com o intuito de deixar claro para o jogador que toda vez que essa tela aparecia, ele estava recebendo uma mensagem.

- **Usos Conhecidos:** Fallout 4 (Usado para o Inventário do jogador), Metro Last Night (Usado para o Inventário do jogador), The Division(Usado para o Mapa do jogador).

### 5.2.2 Dicas de Loading

Figura 9 – Padrão Dicas de Loading sendo aplicado.


Fonte: Própria

- **Nome do Padrão:** Dicas de Loading;
- **Intenção:** Mostrar dicas ao jogador enquanto ele espera o carregamento do jogo terminar;
- **Motivação:** Em vários jogos é necessário que seja feito o carregamento do conteúdo do jogo para que o jogador possa começar a jogar. No entanto, nesse meio tempo o jogador não interage com o jogo de nenhuma forma. Para aliviar isso, durante as telas de carregamento o jogo pode dar dicas que irão ajudar o jogador quando ele for realmente jogar;

- **Aplicabilidade:** Esse padrão deve ser usado quando o jogador precisar esperar o carregamento de algum conteúdo sem que ele interaja com o jogo de nenhuma forma;

- **Consequências:**

  – O jogador pode ganhar conhecimento das mecânicas do jogo enquanto espera;

  – O jogador pode ficar entretido dependendo do conteúdo das dicas.

- **Implementação:** Para implementar esse padrão é necessário se criar uma tela que irá aparecer toda vez que o jogo tiver que carregar uma grande quantidade de conteúdo. Quando essa tela aparecer, dicas que estão armazenadas aparecerão aleatoriamente ao jogador até que o carregamento esteja completo. A figura 9 mostra como esse padrão foi implementado no jogo. Essa tela sempre aparece quando algum tipo de carregamento de recursos é necessário. Ela mostra dicas que tem o intuito de ajudar o jogador a entender as mecânicas do jogo enquanto ele espera o final do carregamento.

- **Usos Conhecidos:** Devil May Cry 5 (Dicas sobre a mecânicas do jogo aparecem no Loading), NBA 2k19 (Durante o Loading há comentários antecipando os acontecimentos da partida), Sekiro (Dicas sobre a mecânicas do jogo aparecem no Loading).

### 5.2.3 Opções

Figura 10 – Padrão de Opções sendo aplicado no jogo


Fonte: Própria

- **Nome do Padrão:** Opções;

- **Intenção:** Mostrar opções para o jogador ajustar o jogo da forma que mais o agradar.

- **Motivação:** Um jogador pode ser qualquer pessoa, dessa forma jogadores tem preferências diferentes de como querem utilizar um jogo. Nesse sentido, é recomendado que jogos disponibilizem uma maneira de jogadores customizarem o jogo da forma que quiserem. Para isso, existe o padrão de interface de opções que disponibiliza um menu onde o jogador pode fazer todas as alterações possíveis de se realizar em um determinado jogo.

- **Aplicabilidade:** Esse padrão deve ser usado quando o jogo apresentar algum tipo de configuração que o jogador possa mudar.

- **Consequências:**

  - O jogador poderá customizar o jogo para se adequar a suas preferências.
  - O jogador saberá onde procurar quando quiser mudar alguma configuração do jogo.

- **Implementação:** Para implementar esse padrão, deve se ter alguma configuração que o jogador possa alterar. Esse menu deve estar em um lugar de fácil acesso ao jogador. Deve se tomar cuidado com a organização das opções quando há um grande número de configurações disponíveis. No Programmer, as opções disponíveis foram reproduzir ou parar a música do jogo e sair do desafio. A figura 10 mostra como esse padrão foi implementado. Ao clicar no ícone de megafone, o som seria parado se a música estivesse tocando ou vice-versa. Além disso, se o jogador clicar no botão sair do desafio, ele irá voltar para a tela de seleção de desafios. O menu de opções é ativado e desativado através do botão em formato de janela posicionado no canto inferior esquerdo da figura.

- **Usos Conhecidos:** God of War (dificuldade, ajuste de som, etc), Total War Three Kingdoms (controles, ajuste de brilho, etc), Pokemon Red (velocidade do texto, animação de batalhas, etc).

### 5.2.4 Tutorial

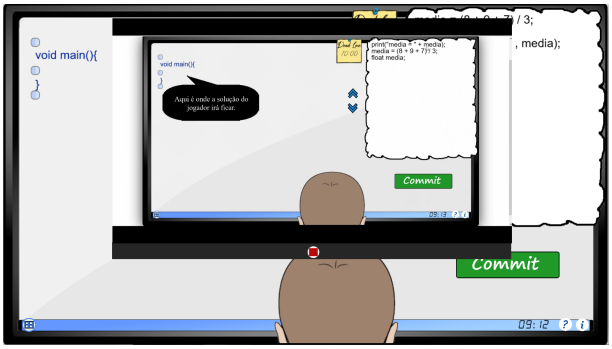
- **Nome do Padrão:** Tutorial;

- **Intenção:** Ajudar o jogador a entender as mecânicas do jogo.

- **Motivação:** Quando um jogo tem muitas mecânicas, o jogador pode se sentir sobrecarregado pela quantidade de informação. Dessa forma, é recomendado que o jogo disponibilize uma forma do jogador se familiarizar com essas mecânicas.

- **Aplicabilidade:** Aplicar quando houver muitas mecânicas em um jogo ou se alguma mecânica for complexa.

Figura 11 – Padrão Tutorial sendo aplicado




Fonte: Própria

- **Consequências:**
  - O jogador terá acesso a explicação das mecânicas.
  - Jogadores mais impacientes podem se frustrar com o tutorial.
- **Implementação:** Para implementar o tutorial o jogo deve disponibilizar uma explicação das mecânicas do dele. Isso pode ser feito de várias formas, como por exemplo o uso de um nível feito para ensinar ao jogador, uso de vídeos demonstrando como jogar, etc. No caso do Programmer, foi implementado um vídeo onde as mecânicas básicas da programação dentro do jogo foram explicadas. Nesse vídeo, a função de cada componente da tela é explicado e uma demonstração de montagem de solução é feita. A figura 11 mostra uma imagem desse vídeo executando. O botão vermelho em baixo do vídeo serve para desativar o tutorial. Já para ativar o tutorial, o jogador pode a qualquer momento pressionar o botão de interrogação no canto inferior direito da tela.
- **Usos Conhecidos:** Cup Head (A primeira fase do jogo ensina os comandos), Total War (Todo o começo do jogo é montado para ensinar as mecânicas ao jogador).

### 5.2.5 Seleção de Fases

- **Nome do Padrão:** Seleção de Fases;
- **Intenção:** Disponibilizar para o jogador a possibilidade de selecionar o nível que quer jogar.
- **Motivação:** Muitos jogos se organizam através de etapas que o jogador tem de cumprir, ou seja, através de níveis. Com isso, é necessário que o jogador possa escolher qual nível ele quer realizar. Para isso, é recomendado que o jogo exiba esses níveis de forma organizada para o jogador.

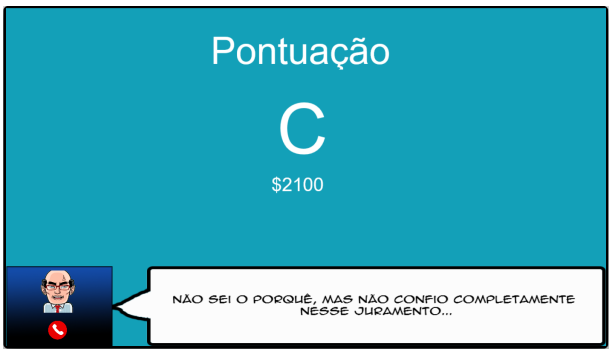
Figura 12 – Padrão Seleção de fases sendo aplicado


Fonte: Própria

- **Aplicabilidade:** Quando o jogo é dividido em níveis que o jogador tem de completar.
- **Consequências:**
  - O jogador poderá escolher qual nível irá fazer.
  - Dependendo da forma com que os níveis foram organizados, o jogador poderá saber sobre a dificuldade dos níveis, temas e etc.
- **Implementação:** Para implementar os níveis, é necessário pegar os níveis presentes dentro do jogo e organiza-los de forma que faça sentido para o jogador. Essa organização pode ser feita de várias formas. No caso do Programmer, os níveis são representados pelos desafios. A seleção de Desafios foi organizada através de grupos de dificuldade e dentro deles os desafios são organizados por ordem alfabética. Além disso, todos os desafios que já foram cumpridos são marcados com um símbolo de "certo", dessa forma o jogador tem noção do seu progresso neles. A figura 12 mostra como esse níveis foram implementados dentro do jogo.
- **Usos Conhecidos:** Super Mario world (A seleção de níveis é feita através de um mapa), Cup Head (A seleção de níveis é feita através de um mapa), Devil May Cry 5 (A seleção de níveis é feita através de uma lista).

### 5.2.6 Tela de Pontuação

Figura 13 – Padrão Tela de Pontuação sendo aplicado




Fonte: Própria

- **Nome do Padrão:** Tela de Pontuação;
- **Intenção:** Mostrar para o jogador seu desempenho ao jogar o jogo.
- **Motivação:** Uma boa forma de se incentivar o jogador a continuar jogando o jogo, é dar pontuação ao seu desempenho, dessa forma ele se sente motivado a melhorar. Nesse sentido, surge a necessidade de se criar uma tela que mostre esse resultado para o jogador.
- **Aplicabilidade:** O jogo tem algum tipo de medição de desempenho do jogador e é necessário mostrar esse isso a ele.
- **Consequências:**
  - O jogador poderá se sentir motivado a melhorar no jogo.
  - O jogaodor pode se sentir frustrado ao não receber uma boa pontuação.
- **Implementação:** Para implementar esse padrão o jogo deve, de alguma forma, calcular o desempenho do jogador e depois o exibir na tela. No caso do Programmer, o desempenho é medido através do tempo que o jogador completa um desafio e quantas vezes ele erra. Em relação ao tempo, quanto menor o tempo levado para completar o desafio, maior a pontuação do jogador. Já no caso da quantidade de erros, quantos mais erros menor a pontuação do jogador. No jogo, dinheiro representa a pontuação e ele pode ser usado para desbloquear níveis mais difíceis, incentivando o jogador a tentar conseguir uma pontuação maior. Além do valor número da pontuação, uma letra foi associada a pontuação para facilitar a visualização do desempenho do jogador. Por exemplo, se o jogador for muito bem em um desafio, ele irá receber uma letra “A”. A figura 13 mostra a tela implementada.

- **Usos Conhecidos:** Devil May cry 5 (O jogador recebe uma pontuação de "estilo" ao final da fase), Cup Head (O jogador recebe uma pontuação baseada na perfomace), Super Mario World (O jogador recebe uma pontuação baseada na perfomace).

## 5.3 Resultado da Validação

Nesta seção os resultados da validação serão apresentados.

### 5.3.1 Resultados dos Professores

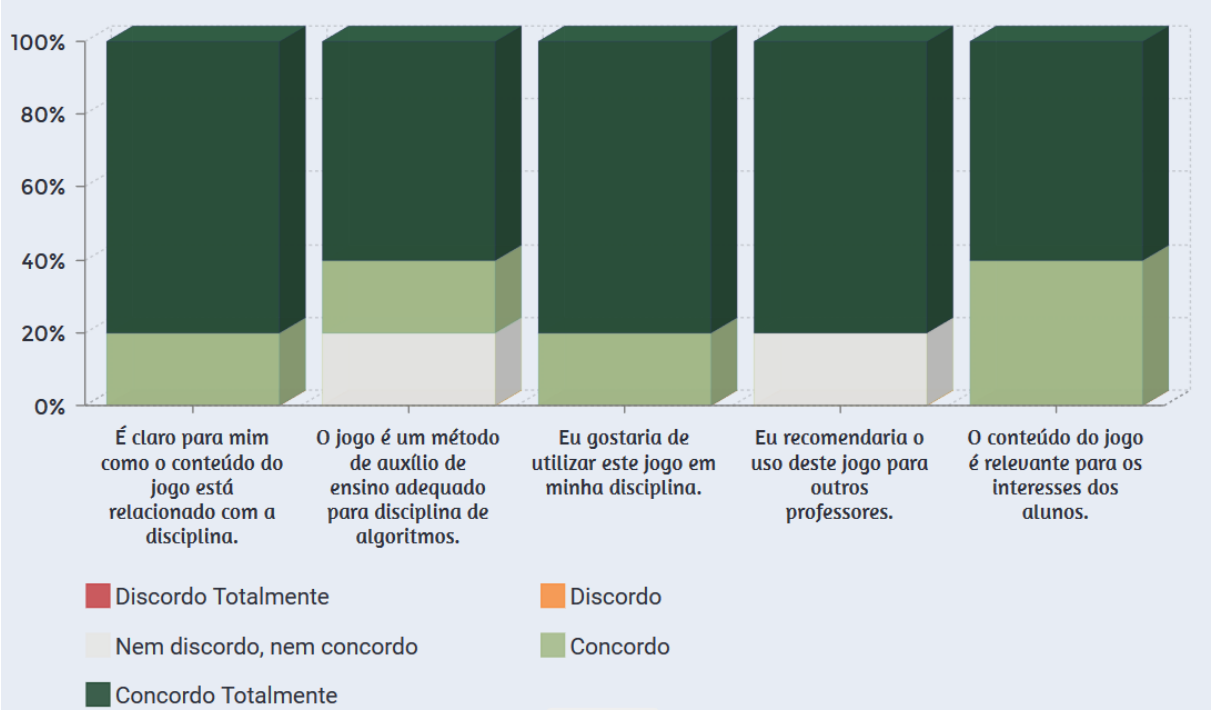
Para saber a opinião de professores de algoritmos a respeito do jogo gerado, um questionário do MEEGA+ foi aplicado. Nesse sentido, um total de 5 professores fizeram a validação do jogo. Todos eles faziam parte da Unipampa e estavam divididos em 3 homens e 2 mulheres. Além disso, um desses professores dá aula no curso de Ciência da Computação, outros 2 dão aula para Engenharia de Software e o resto dá aulas para ambos os cursos. A faixa de idade de todos os professores está entre 29 e 39 anos.

A primeira pergunta do questionário foi "Quantos jogos educacionais (digitais e/ou não-digitais) você já utilizou nas suas aulas (incluindo em outras disciplinas)?". Nessa pergunta, 4 dos professores disseram que utilizaram menos de 5 jogos educacionais em suas aulas. Os outros 1 disseram que nunca utilizaram um jogo educativo em suas aulas. Outra pergunta presente no questionário foi "Você já desenvolveu e/ou customizou jogos educacionais?". Nela, 4 dos professores disseram que não e 1 disseram que sim. Esses dados indicam que os professores não têm muita experiência com jogos educativos.

As próxima parte do questionário tratava de afirmações, em que os professores diziam o quanto eles concordavam com elas através da Escala Likert, em que eles deveriam responder numa escala de 1 a 5. O primeiro grupo de afirmações feitas aos professores lidam com a percepção deles em relação a utilidade do jogo para o ensino de algoritmos. As respostas da Figura 14 indicam que a maioria de professores acharam o conteúdo do jogo relevante para os alunos e que ele poderia ser utilizado para ensinar algoritmos.

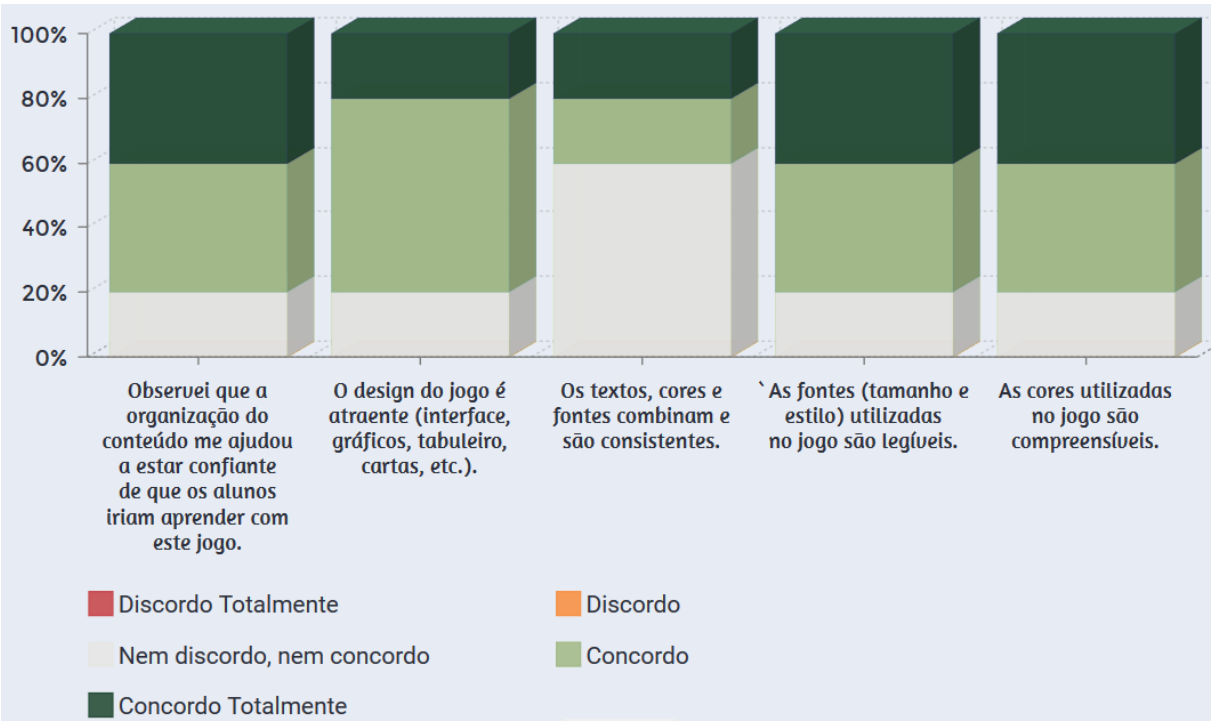
Já o próximo grupo de afirmações, tratam da percepção dos professores sobre a interface do jogo. As respostas dos professores na Figura 15 indicam que, no geral, a interface do jogo é compreensível, no entanto ainda há espaço para melhorar. O último grupo de afirmações, trata da percepção dos professores sobre a usabilidade e facilidade de utilizar o jogo. As respostas dos professores na Figura 16 indicam que, no geral, eles gostaram da usabilidade do jogo. No entanto há, também, aspectos que podem ser melhorados.

Figura 14 – Questões relacionadas ao ensino




Fonte: Própria

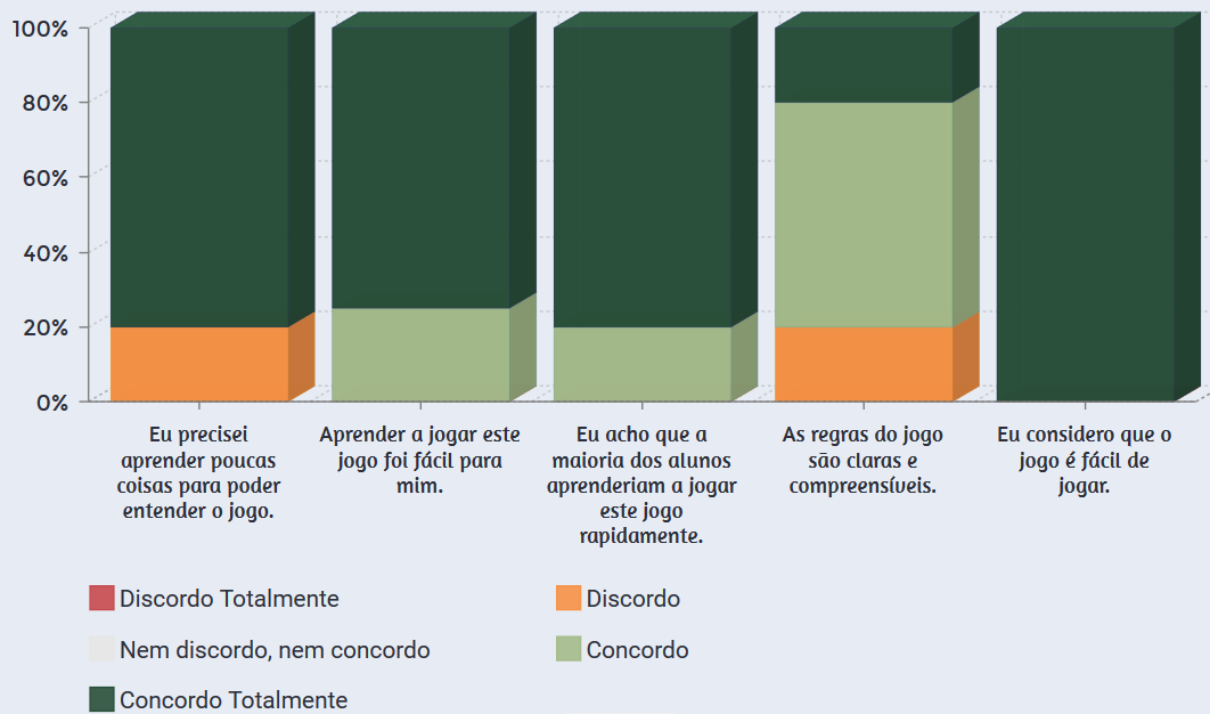
Figura 15 – Questões relacionadas a Interface




Fonte: Própria

Com isso, a próxima parte do questionário pediu aos professores para escreverem três aspectos positivos e sugestões de melhoria para o jogo. Um dos aspectos positivos

Figura 16 – Questões relacionadas a Usabilidade




Fonte: Própria

que os professores observaram foi que o jogo possibilitava o ensino de programação de forma lúdica. Além disso, os professores disseram que o fato do jogo ser parametrizável era um ponto positivo dele e que o jogo era intuitivo para se jogar.

Uma das melhorias que os professores sugeriram foi de que existisse uma forma deles customizarem os desafios. Além disso, eles sugeriram que houvesse um *survey* com os professores, para definir um grupo de desafios padrões para o jogo. Dessa forma os professores que não quisessem customizar o desafio, poderiam utilizar desses desafios padrão. Outra sugestão, foi que os desafios pudessem ser organizados por temas, como por exemplo, sequenciais, com condição, com repetição, etc. Além disso, os professores sugeriram que o enunciado do desafio sempre ficasse na tela, pois era difícil se memorizar o enunciado durante o jogo. Eles também sugeriram que houvesse um *feedback* indicando onde o jogador errou ao enviar uma solução para o servidor. Outra sugestão foi de que poderia ter uma opção livre de digitação de trechos de código, em que os jogadores pudessem escrever a solução. A última sugestão deles foi que o jogo fosse amplamente acessível, fazendo com que pessoas cegas também pudessem joga-lo.

Além dos pontos positivos e das sugestões de melhorias, o pesquisador anoutou algumas observações que os professores fizeram enquanto estavam jogando. Uma delas foi o fato de que o contador de tempo era progressivo e isso não era intuitivo, seria melhor se o tempo fosse regressivo dizendo o tempo total que o jogador têm para resolver um desafio. Além disso, alguns professores disseram que o tutorial poderia ser apresentado

antes do jogador iniciar a parte de programação, pois assim ele saberia das mecânicas antes de experimenta-las. Outra observação foi que o saldo do jogador não era visível o suficiente.

Analisando o *feedback* dos professores, é possível notar que a maioria deles gostaram da ideia do jogo e acharam a maioria dos aspectos do jogo intuitivos. Isso pode ser uma indicação de que os padrões de interface ajudaram os professores a se familiarizar com o jogo. Por exemplo, o padrão *Skeuomorphic* pode ter ajudado os professores com a associação do jogo com a realidade da programação. No entanto, uma das melhorias sugeridas pelos professores foi a mudança do relógio de progressivo para regressivo. A escolha de um relógio progressivo foi feita para continuar respeitando o padrão *Skeuomorphic*, só que isso pode ter acabado influenciando na intuitividade do jogo. Esse fato indica que apesar de ser bom respeitar os padrões, é necessário prestar atenção em outros fatores ao planejar uma interface.

### 5.3.2 Resultados dos Alunos

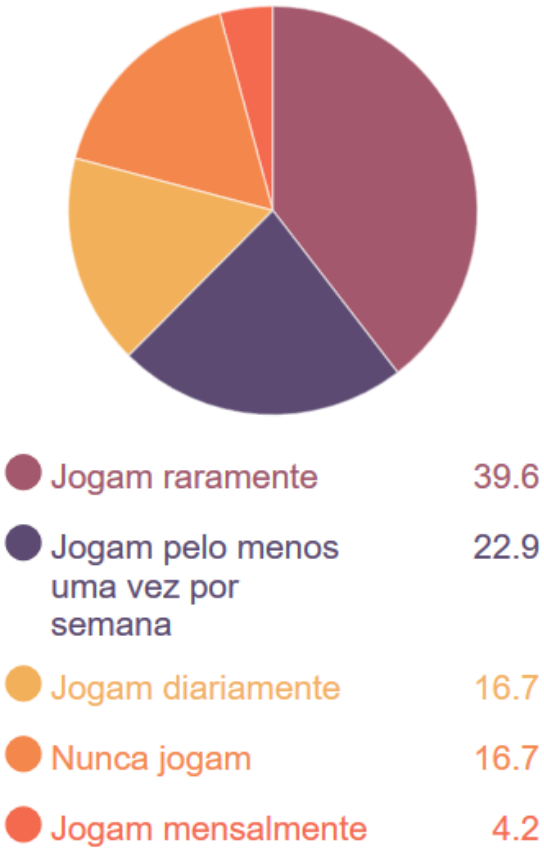
Uma avaliação foi feita com os alunos, na qual os alunos jogaram o jogo e responderam um questionário sobre a opinião deles sobre ele. Todos eram alunos da Unipampa, além disso 66,7% das pessoas eram do sexo masculino e 33,4% eram do sexo feminino. 21 desses alunos pertencem ao curso de Engenharia Agrícola e 27 pertencem ao curso de Engenharia Elétrica. A média de idade mais frequente foi de 18 a 28 anos, com uma porcentagem de 79,2%. 12,5% dos alunos tinham entre 29 e 39 anos, outros 6,3% tinham menos de 18 e 2,1% tinham mais de 50 anos. As Figuras 17 e 18 mostram a frequência que os alunos jogavam jogos digitais e não-digitais (de cartas, tabuleiro, etc.), respectivamente.

As próxima parte do questionário tratava de afirmações, em que os alunos diziam o quanto eles concordavam com elas, através da escala richter. Eles deveriam responder numa escala de 1 a 5, no qual 1 representa discordo totalmente e 5 representa concordo totalmente. A figura 19 mostra os resultados dessa parte, sendo que a coluna "Afirmação" representa a afirmação de um item do questionário e as outras colunas representam a porcentagem de alunos que deram determinada nota. Como por exemplo, 35,4% dos alunos deram nota 5 para a afirmação "O design do jogo é atraente (tabuleiro, cartas, interfaces, gráficos, etc.).".

Analisando os dados respondidos pelos alunos na figura 19, é possível observar que eles tiveram uma impressão positiva sobre a usabilidade do jogo, no entanto o jogo pode melhorar em alguns aspectos, principalmente, na quantidade de mecânicas que os jogadores têm de aprender de uma vez, pois muitos deles tiveram a impressão de que o jogo não foi simples de se aprender.

O próximo grupo de afirmações lidavam com a experiência do jogador. As Figuras 20 e 21 mostram tabelas com os resultados das respostas dos alunos em relação ao suas experiências com o jogo. As tabelas foram organizadas da mesma forma que a tabela

Figura 17 – Frequência com que os alunos jogavam jogos digitais.




Fonte: Própria

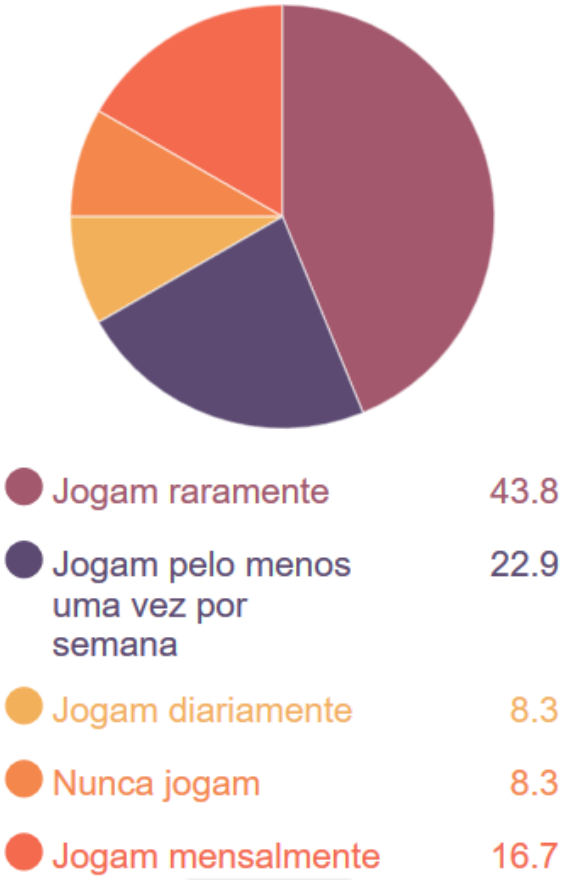
anterior.

Ao analisar os dados é possível perceber que a maioria dos alunos teve uma boa experiência com o jogo: a maioria percebeu a associação do jogo com o conteúdo que eles estavam vendo em algoritmos, a maioria achou que o jogo pode contribuir para para a aprendizagem e a maioria se divertiu com o jogo. É importante destacar que os alunos que deram notas mais baixas a sua experiência, foram os alunos que tinham menos experiência ou não gostavam de jogos.

Além das afirmações, houve três perguntas para os alunos escreverem suas opiniões. A primeira foi "O que você mais gostou no jogo?". Nela os alunos disseram que gostaram dos desafios apresentados, das mecânicas, das dinâmicas do jogo e o fato do jogo ajudar no aprendizado da disciplina. Já a segunda pergunta era "O que poderia ser melhorado no jogo?", em que os alunos disseram que o tempo para resolver os desafios poderia ser maior, poderia ter mais desafios para se resolver, colocar mais dicas e melhorar a responsividade do jogo para tela menores. A terceira e última pergunta dizia "Gostaria de fazer mais algum comentário?", na qual os alunos elogiaram a criação do jogo e os outros não quiseram colocar nenhum comentário adicional.

Analisando as impressões que os alunos tiveram do jogo, nota-se que a maioria

Figura 18 – Frequência com que os alunos jogavam jogos não-digitais.




Fonte: Própria

gostou do jogo e achou ele adequado para a aprendizagem. Além disso. os alunos mais experientes com jogos tiveram menos dificuldade de joga-lo. Isso pode ser uma indicação que eles reconheceram os padrões aplicados e isso fez com que eles entendessem as mecânicas do jogo de forma mais fácil.

A maioria dos professores e alunos concordaram na possível utilidade do jogo como ferramenta educativa. Ambos os grupos reclamaram do tempo disponível para se resolver um desafio proposto. No entanto, os professores deram mais sugestões de melhoria relacionadas a interface. Isso pode ter acontecido, devido ao conhecimento técnico que os professores têm.

Figura 19 – Tabela mostrando as respostas dos alunos relacionadas a usabilidade.

| Afirmação | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| O design do jogo é atraente (tabuleiro, cartas, interfaces, gráficos, etc.). | 0% | 2,1% | 29,2% | 33,3% | 35,4% |
| Os textos, cores e fontes combinam e são consistentes. | 0% | 0% | 18,8% | 16,7% | 64,6% |
| Eu precisei aprender poucas coisas para poder começar a jogar o jogo. | 4,2% | 14,6% | 31,3% | 22,9% | 27,1% |
| Aprender a jogar este jogo foi fácil para mim. | 8,3% | 8,3% | 12,5% | 31,3% | 39,6% |
| Eu acho que a maioria das pessoas aprenderiam a jogar este jogo rapidamente. | 12,5% | 14,6% | 29,2% | 16,7% | 27,1% |
| Eu considero que o jogo é fácil de jogar. | 2,1% | 22,9% | 22,9% | 20,8% | 31,3% |

Fonte: Própria

Figura 20 – Tabela mostrando as respostas dos alunos relacionadas a experiência.

| Afirmação | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Este jogo é adequadamente desafiador para mim. | 2.1% | 6,3% | 16,7% | 35,4% | 39,6% |
| O jogo oferece novos desafios (oferece novos obstáculos, situações ou variações) com um ritmo adequado. | 0% | 2,1% | 18,8% | 39,6% | 39,6% |
| `O jogo não se torna monótono nas suas tarefas (repetitivo ou com tarefas chatas). | 2,1% | 12,5% | 14,6% | 22,9% | 47,9% |
| Completar as tarefas do jogo me deu um sentimento de realização. | 0% | 6,3% | 33,3% | 25% | 35,4% |
| É devido ao meu esforço pessoal que eu consigo avançar no jogo. | 0% | 4,2% | 14,6% | 33,3% | 47,9% |
| Me sinto satisfeito com as coisas que aprendi no jogo. | 0% | 2,1% | 14,6% | 39,6% | 43,8% |
| Eu recomendaria este jogo para meus colegas. | 0% | 2,1% | 16,7% | 18,8% | 62,5% |
| Eu pude interagir com outras pessoas durante o jogo. | 14,6% | 6,3% | 14,6% | 20,8% | 43,8% |
| O jogo promove momentos de cooperação e/ou competição entre os jogadores. | 10,4% | 12,5% | 12,5% | 22,9% | 41,7% |
| O jogo foi eficiente para minha aprendizagem, em comparação com outras atividades da disciplina. | 2,1% | 4,2% | 20,8% | 31,3% | 41,7% |
| O jogo contribuiu para relembrar os conceitos de algoritmos | 0% | 0% | 10,4% | 27,1% | 62,5% |

Fonte: Própria

Figura 21 – Tabela mostrando as respostas dos alunos relacionadas a experiência.

| Afirmação | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|
| Eu me senti bem interagindo com outras pessoas durante o jogo. | 16,7% | 2,1% | 29,2% | 14,6% | 37,5% |
| Eu me diverti com o jogo. | 2,1% | 6,3% | 20,8% | 22,9% | 47,9% |
| Aconteceu alguma situação durante o jogo (elementos do jogo, competição, etc.) que me fez sorrir | 10,4% | 18,8% | 18,8% | 20,8% | 31,3% |
| Houve algo interessante no início do jogo que capturou minha atenção. | 2,1% | 18,8% | 14,6% | 27,1% | 37,5% |
| Eu estava tão envolvido no jogo que eu perdi a noção do tempo. | 6,3% | 12,5% | 25% | 22,9% | 33,3% |
| Eu esqueci sobre o ambiente ao meu redor enquanto jogava este jogo. | 6,3% | 18,8% | 27,1% | 22,9% | 25% |
| O conteúdo do jogo é relevante para os meus interesses. | 4,2% | 4,2% | 6,3% | 25% | 60,4% |
| É claro para mim como o conteúdo do jogo está relacionado com a disciplina. | 0% | 2,1% | 0% | 14,6% | 83,3% |
| O jogo é um método de ensino adequado para esta disciplina. | 0% | 0% | 6,3% | 18,8% | 75% |
| Eu prefiro aprender com este jogo do que de outra forma (outro método de ensino). | 8,3% | 4,2% | 22,9% | 31,3% | 33,3% |
| O jogo contribuiu para a minha aprendizagem na disciplina. | 0% | 4,2% | 12,5% | 16,7% | 66,7% |

Fonte: Própria

# 6 CONSIDERAÇÕES FINAIS

Jogos estão sendo muito popularizados nos últimos anos, e com isso cresceu o interesse dos desenvolvedores em aplicar padrões para interfaces em jogos, de forma a aumentar produtividade da equipe e reusar aprendizado prévio dos jogadores. No entanto, ainda não existe uma catalogação adequada desses padrões, da forma que este trabalho propôs a coleta e catalogação de padrões de interface para jogos. Além disso, a criação de um jogo foi proposta, a fim de servir de exemplo de implementação e mostrar os padrões sendo utilizados. A partir disso, o estilo desse jogo foi determinado como educativo, pois o jogo final poderia ser utilizado para o ensino na universidade onde foi desenvolvido, além de ser uma prova de conceito para os padrões. Ainda, o tema a ser ensinado pelo jogo foi programação, pois muitos alunos têm dificuldade com o tema ao ingressar na faculdade.

Esse trabalho criou um jogo chamado *Programmer*, feito através da aplicação de uma lista de padrões catalogados. A partir do próprio catálogo, uma pessoa que for inexperiente na área de jogos pode utilizar essa lista de padrões de interface como referência e pode vê-los aplicados no jogo criado. Além disso, o *Programmer* foi feito com o ensino de algoritmos em mente, portanto o jogo pode ser usado por professores para auxiliar no ensino da disciplina.

Como limitações desse estudo, destaca-se que o jogo não contém todos os padrões de interface encontrados, pois alguns deles não se aplicariam a plataforma de jogo que foi proposto. Além disso, através do *feedback* da avaliação, nota-se que há aspectos do jogo que podem ser melhorados, como por exemplo, sempre mostrar o enunciado do desafio.

Como trabalhos futuros, é viável propor novos jogos para se aplicar padrões de interface que não foram tratados por esse trabalho ou expandir esse para novos modos de jogo que consigam incluir esses padrões. Além disso, pode-se citar que é possível implementar os *feedbacks* adquiridos através dos alunos e professores, como por exemplo: responsividade para telas pequenas, enunciado sempre visível, mais métodos de classificação de desafios e etc. Com isso, o Programmer poderá evoluir e melhorar o processo de aprendizado dos alunos.

# REFERÊNCIAS

ADAM, D. **Game Patterns**. 2013. Disponível em: <http://www.game-patterns.com>. Citado na página 30.

ADAMS, R. E. M. D. M.; MACNAMARA, A. K. A.; Richard Wainess. Narrative Games for Learning: Testing the Discovery and Narrative Hypotheses. **Journal of Educational Psychology**, v. 104, 2011. Citado 2 vezes nas páginas 11 e 24.

ALEXANDER. **Game Ui Patterns**. 2009. Disponível em: <https://www.flickr.com/photos/headorange/collections/72157621971763357>. Citado na página 30.

BARENDREGT, T. M. B. W. The influence of the level of free-choice learning activities on the use of an educational computer game. **Computers & Education**, v. 56, n. 1, 2010. Citado na página 11.

CHAU, B. et al. Corrupted: A game to teach programming concepts. **Computer**, IEEE, v. 47, n. 12, p. 100–103, 2014. Citado na página 23.

COPLIEN, J. O.; ALEXANDER, A. W. O. Software patterns. Citeseer, 1996. Citado 2 vezes nas páginas 18 e 19.

CRAWFORD, C. The art of computer game design. Osborne/McGraw-Hill Berkeley, CA, 1984. Citado na página 15.

DICKEY, M. D. Engaging by design: How engagement strategies in popular computer and video games can inform instructional design. **Educational Technology Research and Development**, Springer, v. 53, n. 2, p. 67–83, 2005. Citado na página 25.

DIVJAK, B.; TOMIĆ, D. The impact of game-based learning on the achievement of learning goals and motivation for learning mathematics-literature review. **Journal of Information and Organizational Sciences**, Fakultet organizacije i informatike Sveučilišta u Zagrebu, v. 35, n. 1, p. 15–30, 2011. Citado na página 15.

EAGLE, M.; BARNES, T. Wu's castle: teaching arrays and loops in a game. In: ACM. **ACM SIGCSE Bulletin**. [S.l.], 2008. v. 40, n. 3, p. 245–249. Citado na página 23.

EBNER, M.; HOLZINGER, A. Successful implementation of user-centered game based learning in higher education: An example from civil engineering. **Computers & education**, Elsevier, v. 49, n. 3, p. 873–890, 2007. Citado na página 15.

FAGERHOLT, E.; LORENTZON, M. Beyond the hud-user interfaces for increased player immersion in fps games. Chalmers University of Technology, 2009. Citado 3 vezes nas páginas 26, 27 e 30.

FANG, F. Z. X. Personality and enjoyment of computer game play. **Computers in Industry**, 2010. Citado na página 11.

FOLMER, E. Usability patterns in games. **Future Play**, v. 6, 2006. Citado 3 vezes nas páginas 25, 26 e 30.

FONTES, C. R.; SILVA, F. W. da. O ensino da disciplina linguagem de programação em escolas técnicas. **Ciências & Cognição**, Ciências e Cognição, n. 13, p. 84–98, 2008. Citado na página 14.

FRAZER, A. et al. Profiling the educational value of computer games. **Interaction Design & Architecture (s) Journal-IxD&A**, v. 19, p. 1–19, 2014. Citado na página 15.

GAMMA, E. **Padrões de Projetos: Soluções Reutilizáveis**. [S.l.]: Bookman, 2007. Citado 2 vezes nas páginas 31 e 67.

GAMMA, E. et al. **Design patterns: elements of reusable object-oriented software**. [S.l.]: Pearson Education, 1994. Citado na página 18.

HERNANDEZ, L. S. C. C. et al. Teaching Programming Principles through a Game Engine. **CLEI ELECTRONIC JOURNAL**, v. 13, n. 2, 2010. Citado 2 vezes nas páginas 11 e 23.

JÚNIOR, J. C. R. P.; RAPKIEWICZ, C. E. O processo de ensino-aprendizagem de fundamentos de programação: Uma visão crítica da pesquisa no brasil. In: **WEI-Workshop sobre Educação em Computação**. [S.l.: s.n.], 2004. p. 19–21. Citado na página 13.

MCGONIGAL, J. A realidade em jogo: por que os games nos tornam melhores e como eles podem mudar o mundo. **Rio de Janeiro: Bestseller**, 2012. Citado na página 14.

MORENO-GER, P. et al. Educational game design for online education. **Computers in Human Behavior**, Elsevier, v. 24, n. 6, p. 2530–2540, 2008. Citado na página 17.

NEIL, T. **Mobile Design Pattern Gallery: UI Patterns for Smartphone Apps**. [S.l.]: "O'Reilly Media, Inc.", 2014. Citado na página 20.

PAPASTERGIOU, M. Digital game-based learning in high school computer science education: Impact on educational effectiveness and student motivation. **Computers & Education**, Elsevier, v. 52, n. 1, p. 1–12, 2009. Citado na página 15.

PETRILLO, F. Práticas ágeis no processo de desenvolvimento de jogos eletrônicos. **UFRGS. Instituto de Informática. Porto Alegre–RS**, 2008. Citado na página 16.

PRENSKY, M. Computer games and learning: Digital game-based learning. **Handbook of computer game studies**, MIT Press Cambridge, MA, v. 18, p. 97–122, 2005. Citado na página 15.

PRESSMAN, R. S. **Engenharia de software**. [S.l.]: McGraw Hill Brasil, 2011. Citado na página 19.

RANKIN, Y.; GOOCH, A.; GOOCH, B. The impact of game design on students' interest in cs. In: ACM. **Proceedings of the 3rd international conference on Game development in computer science education**. [S.l.], 2008. p. 31–35. Citado na página 24.

RIPASY, R. et al. Assistant meega+: Uma ferramenta de apoio para avaliação de jogos educacionais usando modelo meega+. In: **Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação-SBIE)**. [S.l.: s.n.], 2018. v. 29, n. 1, p. 615. Citado na página 31.

ROBINS, J. R. A.; ROUNTREE, N. Learning and Teaching Programming: A Review an Discussion. **Computer Science Education**, v. 13, n. 2, 2003. Citado 2 vezes nas páginas 11 e 13.

ROGALSKI, J.; SAMURÇAY, R. Acquisition of programming knowledge and skills. **Psychology of programming**, Academic Press London, England, v. 18, p. 157–174, 1990. Citado na página 13.

SCHELL, J. **The Art of Game Design: A book of lenses**. [S.l.]: CRC Press, 2014. Citado na página 14.

SCHWABER, K. Scrum development process. In: **Business Object Design and Implementation**. [S.l.]: Springer, 1997. p. 117–134. Citado na página 29.

TANG, V. **Games UI**. 2015. Disponível em: <http://gamesui.com>. Citado na página 30.

# Apêndices

# APÊNDICE A – PRODUCT BACKLOG NO TRELLO

Figura 22 – Cartões do Trello contendo as tarefas que foram realizadas do Product Backlog


Fonte: Própria

# Anexos

# ANEXO A – ESTRUTURA DO PADRÃO

Existem várias formas para se organizar a estrutura de um padrão. A estrutura demonstrada aqui foi proposta por Gamma (2007). Nesse sentido, a estrutura dos padrões tem alguns elementos principais:

- **Nome do Padrão:** Aqui é descrito o nome do padrão;
- **Intenção:** Aqui é especificado qual a é a intenção do padrão para se usar esse padrão;
- **Motivo:** Aqui se explica a motivação para se usar o padrão;
- **Aplicabilidade:** Aqui se explica quando se utilizar o padrão;
- **Estrutura:** Aqui se demonstra qual a estrutura do padrão, ou seja, é feita sua representação em uma modelagem de classes.
- **Participantes:** Aqui as classes demonstradas na estrutura são explicadas.
- **Colaborações:** Aqui os relacionamentos entre as classes são especificados.
- **Conseqüências:** Aqui se mostra quais são as consequências de se aplicar o padrão, incluindo benefícios e malefícios;
- **Implementação:** Aqui se mostra como é feita a implementação desse padrão;
- **Exemplos:** Aqui se mostra exemplos do padrão sendo aplicado;
- **Usos Conhecidos:** Aqui se mostra quais os produtos que já utilizam esse padrão;
- **Padrões relacionados:** Se existirem outros padrões relacionados a este, eles devem ser citados aqui.

O Gamma (2007) apresenta um exemplo de padrão organizado nesse formato:

- **Nome do Padrão:** *SINGLETON*;
- **Intenção:** Garantir que uma classe tenha somente uma instância e fornecer um ponto global de acesso para a mesma;
- **Motivo:** É importante para algumas classes ter uma, e apenas uma, instância. Por exemplo, embora possam existir muitas impressoras em um sistema, deveria haver somente um *spooler* de impressoras. Da mesma forma, deveria haver somente um sistema de arquivos e um gerenciador de janelas. Um filtro digital terá somente um conversor A/ D. Um sistema de contabilidade será dedicado a servir somente a uma companhia. Como garantimos que uma classe tenha somente uma instância e que

essa instância seja facilmente acessível? Uma variável global torna um objeto acessível, mas não impede você de instanciar múltiplos objetos. Uma solução melhor seria tornar a própria classe responsável por manter o controle da sua única instância. A classe pode garantir que nenhuma outra instância seja criada (pela interceptação das solicitações para criação de novos objetos), bem como pode fornecer um meio para acessar sua única instância. Este é o padrão *SINGLETON*;

- **Aplicabilidade:** Use o padrão Singleton quando: for preciso haver apenas uma instância de uma classe, e essa instância tiver que dar acesso aos clientes através de um ponto bem conhecido; a única instância tiver de ser extensível através de subclasses, possibilitando aos clientes usar uma instância estendida sem alterar o seu código.

- **Participantes:**

  - Sigleton
    - define uma operação *Instance* que permite aos clientes acessarem sua única instância. *Instance* é uma operação de classe (ou seja, em *Smalltalk* é um método de classe e em C++ é uma função-membro estática).
    - pode ser responsável pela criação da sua própria instância única.

- **Estrutura:**

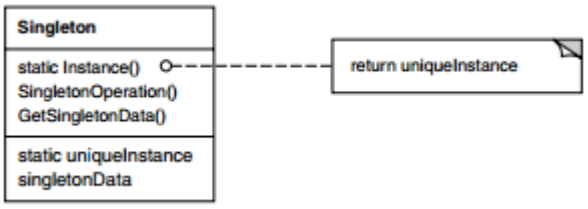



Figura 23 – Estrutura do *Singleton*

- **Colaborações:** Os clientes acessam uma instância *Singleton* unicamente pela operação *Instance* do *Singleton*.

- **Consequências:**

  - **Acesso controlado à instância única:** Como a classe *Singleton* encapsula a sua única instância, possui controle total sobre como e quando os clientes a acessam;
  - **Espaço de nomes reduzido:** O padrão *Singleton* representa uma melhoria em relação ao uso de variáveis globais. Ele evita a poluição do espaço de nomes com variáveis globais que armazenam instâncias únicas;

- **Permite um refinamento de operações e da representação:** A classe *Singleton* pode ter subclasses e é fácil configurar uma aplicação com uma instância dessa classe estendida. Você pode configurar a aplicação com uma instância da classe de que necessita em tempo de execução;
- **Permite um número variável de instâncias:** O padrão torna fácil mudar de idéia, permitindo mais de uma instância da classe *Singleton*. Além disso, você pode usar a mesma abordagem para controlar o número de instâncias que a aplicação utiliza. Somente a operação que permite acesso à instância de *Singleton* necessita ser mudada; Mais flexível do que operações de classe. Uma outra maneira de empacotar a funcionalidade de um *Singleton* é usando operações de classe (ou seja, funções-membro estáticas em C++ ou métodos de classe em *Smalltalk*). Porém, as técnicas de ambas as linguagens tornam difícil mudar um projeto para permitir mais que uma instância de uma classe. Além disso, as funções-membro estáticas em C++ nunca são virtuais, o que significa que as subclasses não podem redefini-las polimorficamente.

- **Implementação:**

1. **Garantindo uma única instância:** O padrão Singleton torna a instância única uma instância normal de uma classe, mas essa classe é escrita de maneira que somente uma instância possa ser criada. Uma forma comum de fazer isso é ocultando a operação que cria a instância usando uma operação de classe (isto é, ou uma função-membro estática ou um método de classe) que garanta que apenas uma única instância seja criada. Esta operação tem acesso à variável que mantém a única instância, e garante que a variável seja iniciada com a instância única antes de retornar ao seu valor. Esta abordagem assegura que um singleton seja criado e iniciado antes da sua primeira utilização. Em C++, você pode definir a operação de classe com uma função-membro estática Instance da classe Singleton. Singleton também define uma variável-membro estática instance que contém um apontador para sua única instância. A classe Singleton é declarada como:

```
class Singleton {
public:
        static Singleton* Instace();
protected:
        Singleton{};
private
        static Singleton* instance;
};
```

A implementação correspondente é:

```
Singleton* Singleton:: instance = 0;

Singleton* Singleton::Instance () {
        if(instance == 0) {
                instance = new Singleton;
        }
        return instance;
}
```

Os clientes acessam o singleton através da função membro Instance. A variável instance é iniciada com 0, e a função-membro estática Instance retorna o seu valor, iniciando-a com a única instância se ele for 0. Instance usa lazy initialization; o valor que ela retorna não é criado e armazenado até ser acessado pela primeira vez. Note que o constructor é protegido. Um cliente que tenta instanciar Singleton diretamente obterá como resposta um erro em tempo de compilação. Isto assegura que somente uma instância possa ser criada. Além do mais, uma vez que instance é um apontador para um objeto Singleton, a função-membro Instance pode atribuir um apontador para uma subclasse de Singleton para esta variável. Daremos um exemplo do que dissemos aqui na seção Exemplo de código. Há uma outra coisa a ser observada sobre a implementação em C++. Não é suficiente definir o singleton como um objeto global ou estático, confiando numa inicialização automática. Existem três razões para isto:

- não podemos garantir que somente uma instância de um objeto estático será declarada;
- em tempo de inicialização estática. Um singleton pode necessitar de valores que são computados mais tarde, durante a execução do programa;
- C++ não define a ordem pela qual os constructors para objetos globais são chamados entre unidades de compilação. Isso significa que não podem existir dependências entre singletons; se alguma existir, então é inevitável a ocorrência de erro.

Uma deficiência adicional (embora pequena) da abordagem objeto global/ estático é que ela força a criação de todos singletons, quer sejam usados ou não. O uso de uma função-membro estática evita todos estes problemas. Em Smalltalk, a função que retorna a instância única é implementada como um método de classe da classe Singleton. Para garantir que somente uma instância seja criada, redefine-se a operação new. A classe Singleton resultante pode ter os

seguintes métodos de classe, em que SoleInstance é uma variável de classe que não é usada em nenhum outro lugar:

```
new
        self error: 'cannot create new object'
default
        SoleInstace isNil ifTrue: [SoleInstance := super new].
```

2. **Criando subclasses da classe Singleton:** O ponto principal não é a definição da subclasse, mas sim a instalação da sua única instância de maneira que os clientes possam ser capazes de usá-la. Em essência, a variável que referencia a instância do singleton deve ser iniciada com uma instância da subclasse. A técnica mais simples é determinar qual singleton você quer usar na operação Instance do Singleton. Um exemplo na seção de Exemplo mostra como implementar essa técnica com variáveis do ambiente (operacional). Uma outra maneira de escolher a subclasse de Singleton é retirar a implementação de Instance da classe-mãe (por exemplo, MazeFactory) e colocá-la na subclasse. Isto permite a um programador C++ decidir a classe do singleton em tempo de "Linkedição" (link-time), mantendo-a oculta dos seus clientes (por exemplo, fazendo a ligação com um arquivo-objeto que contém uma implementação diferente). A solução da ligação fixa a escolha da classe do singleton em tempo de "linkedição", o que torna difícil escolher a classe do singleton em tempo de execução. O uso de instruções condicionais para determinação da subclasse é mais flexível, porém codifica de maneira rígida o conjunto das classes Singleton possíveis. Nenhuma abordagem é flexível o bastante em todos os casos. Uma abordagem mais flexível utiliza um sistema de registro de singletons (registry of singletons). Em vez de ter Instance definindo o conjunto das classes Singleton possíveis, as classes Singleton podem registrar suas instâncias singleton por nome, num sistema de registro de conhecimento geral. O sistema de registro associa nomes e singletons. Os nomes são constituídos de cadeias de caracteres. Quando Instance necessita um singleton, ela consulta o sistema de registro, procurando o singleton pelo nome. O sistema de registro procura o singleton correspondente (se existir) e o retorna ao cliente. Essa solução libera Instance da necessidade de ter que conhecer todas as possíveis classes ou instâncias do Singleton. Tudo o que é necessário é uma interface comum para todas as classes Singleton, que inclua operações de registro:

```
class Singleton {
public:
        static void Register(const char* name, Singleton*);
        static Singleton* Instance();
```

```
protected:
        static Singleton* Lookup(const char* name);
private:
        static Singleton* instance;
        static List<NameSingletonPair>* registry;
}
```

Register registra a instância de Singleton com um nome fornecido. Para manter o registro simples, necessitaremos que armazene uma lista de objetos NameSingletonPair. Cada NameSingletonPair mapeia (associa) um nome a um singleton. Dado um nome, a operação Lookup encontra o singleton correspondente. Assumiremos que uma variável do ambiente especifica o nome do singleton desejado.

```
Singleton* Singleton::Instance () {
        if(instace == 0) {
                const char* singletonName;
                singletonName = getenv("SINGLETON");
                instance = Lookup(singletonName);
        }
        return instance;
}
```

Onde as classes Singleton registram a si mesmas? Uma possibilidade é fazêlo no seu constructor. Por exemplo, uma subclasse MySingleton poderia fazer o seguinte:

```
MySingleton::MySingleton() {
        /...
        Singleton::Register("MySingleton,this");
}
```

Naturalmente, o construtor não será chamado a menos que alguém instancie a classe, o que repete o problema que o padrão Singleton está tentando resolver! Nós podemos contornar este problema em C++ através da definição de uma instância estática de MySingleton. Por exemplo, podemos definir:

```
static MySingleton theSingleton;
```

no arquivo que contém a implementação de MySingleton. A classe Singleton não é mais responsável pela criação do singleton. Em vez disso, sua responsabilidade primária é tornar acessível o objeto singleton escolhido no sistema. A solução que usa o objeto estático ainda apresenta um problema potencial

– todas as instâncias de todas as subclasses possíveis de Singleton devem ser criadas, pois, caso contrário, não serão registradas.

- **Exemplos de código:** Suponha que definimos uma classe MazeFactory para construir labirintos, conforme descrito na página 100. MazeFactory define uma interface para construção de diferentes partes de um labirinto. As subclasses podem redefinir as operações para retornar instâncias de classes-produtos especializadas, tal como BombedWall no lugar de simples objetos Wall. O fato relevante aqui é que a aplicação Maze necessita somente de uma instância de uma fábrica de labirintos, e que essa instância deverá estar disponível para o código que construir qualquer parte do labirinto. É aí que o padrão Singleton entra. Ao tornar MazeFactory um singleton, nós tornamos o objeto-labirinto (maze) acessível globalmente sem recorrer a variáveis globais. Para simplificar, suponhamos que nunca criaremos subclasses de MazeFactory (a alternativa será considerada mais à frente). Nós tornamos MazeFactory uma classe Singleton em C++, acrescentando uma operação estática Instance e um membro estático instance para conter a única instância existente. Também devemos proteger o constructor para prevenir instanciações acidentais, as quais nos levariam a ter mais que uma instância.

```
class MazeFactory {
public:
        static MazeFactory* Instance();
        //interface existente vai aqui;
protected:
        MazeFactory();
private:
        static MazeFactory* instance;
};
```

A implementação correspondente é:

```
MazeFactory* MazeFactory::instance = C

MazeFactory* MazeFactory::Instance () {
        if (instance == 0) {
                instance = new MazeFactory;
        }
        return instance;
}
```

Agora verificaremos o que acontece quando existem subclasses de MazeFactory e a aplicação tem que decidir qual delas usar. Selecionaremos o tipo de labirinto

através de uma variável do ambiente e acrescentaremos o código que instancia a subclasse apropriada de MazeFactory com base no valor da variável do ambiente. Um bom lugar para colocar este código é a operação Instance, porque ela já instancia MazeFactory:

```
MazeFactory* MazeFactory::Instance (){
        if (_instance == 0) {
                const char* mazeStyle = getenv("MAZESTYLE");

                if(strcmp(mazeStyle, "bombed") == 0) {
                        instace = new BombedMazeFactory;
                } else if (strcmp(mazeStyle, "enchanted") == 0){
                        instance = new EnchantedMazeFactory;
                }else {
                        instance = new MazeFactory;
                }
        }
        return instance;
}
```

Note que Instance deve ser modificada toda vez que você define uma nova subclasse de MazeFactory. Isso pode não ser um problema nesta aplicação, mas pode ser um problema para as fábricas abstratas definidas num framework. Uma solução possível seria usar a técnica do uso de um sistema de registro descrita na seção Implementação. A ligação dinâmica (dynamic linking) poderia também ser útil aqui – ela evitaria que a aplicação tivesse que carregar para a memória todas as subclasses que não são usadas.

- **Usos Conhecidos:** Um exemplo do padrão Singleton em Smalltalk-80 [Par90] é o conjunto de mudanças no código efetuado por ChangeSet current. Um exemplo mais sutil é o relacionamento entre classes e suas metaclasses. Uma metaclasse é a classe de uma classe, e cada metaclasse tem uma instância. As metaclasses não têm nomes (exceto indiretamente, através do nome da sua única instância), mas registram e acompanham a sua única instância, e normalmente não criarão outra. O toolkit para construção de interfaces de usuário InterViews [LCI+92] usa o padrão Singleton para acessar as únicas instâncias de suas classes Session e WidgetKit, entre outras. Session define o ciclo de eventos disparáveis da aplicação principal, armazena o banco de dados das preferências de estilo do usuário e administra conexões para um ou mais dispositivos físicos de display. WidgetKit é uma Abstract Factory (95) para definir os widgets de estilo de interação. A operação WidgetKit::instance determina a subclasse específica de WidgetKit que é instanciada baseada numa variável de

ambiente que Session define. Uma operação similar em Session determina se são suportados displays monocromáticos ou coloridos e configura a instância singleton de Session de acordo.

- **Padrões relacionados:** Muitos padrões podem ser implementados usando Singleton. Ver Abstract Factory, Builder e Prototype.