UNIVERSIDADE FEDERAL DE SANTA CATARINA
CAMPUS FLORIANÓPOLIS ou CENTRO TECNOLÓGICO
DEPARTAMENTO DE INFORMÁTICA E ESTATÍSTICA
CURSO SISTEMAS DE INFORMAÇÃO

Igor Glatz

Desenvolvimento de um Jogo para Auxílio no Ensino de Estruturas de Dados

Florianópolis
2023

Igor Glatz

Desenvolvimento de um Jogo para Auxílio no Ensino de Estruturas de Dados

Trabalho Conclusão do Curso de Graduação em Sistemas de Informação do Centro Tecnológico da Universidade Federal de Santa Catarina como requisito para a obtenção do título de Bacharel/Licenciado em Sistemas de Informação.
Orientador: Prof. José Eduardo De Lucca

Florianópolis
2023

Igor Glatz

Desenvolvimento de um Jogo para Auxílio no Ensino de Estruturas de Dados

Este Trabalho Conclusão de Curso foi julgado adequado para obtenção do Título de “Desenvolvimento de um Jogo para Auxílio no Ensino de Estruturas de Dados” e aprovado em sua forma final pelo Curso Sistemas de Informação

Florianópolis, 11 de Julho de 2023.

_____
Prof. José Eduardo De Lucca
Coordenador do Curso

Banca Examinadora:

_____
Prof. José Eduardo De Lucca,
Orientador
Instituição Universidade Federal de Santa Catarina

_____
Prof. Jean Carlo Rossa Hauck, Dr.
Avaliador
Instituição Universidade Federal de Santa Catarina

_____
Profª. Fabiane Barreto Vavassori Benitti, Dra.
Avaliadora
Instituição Universidade Federal de Santa Catarina

Este trabalho é dedicado aos meus pais, amigos e professores que me ajudaram durante todo o processo.

RESUMO

Em um mundo cada vez mais digital, com a incremental importância de dispositivos eletrônicos no nosso dia a dia, se vê necessária a formação de profissionais da área de Tecnologia de Informações, para realizar funções de manutenção e inovação relacionada a esses dispositivos. Um dos caminhos para obter essa formação se dá pela adesão no meio Universitário, na entrada para um curso de Ensino Superior. Porém, nem sempre essa adesão ocorre sem solavancos, sendo o tema de Pensamento Computacional, um tema que, por experiência própria, tem seus detalhes e nuances que dificultam essa aprendizagem. Alguns pilares do Pensamento Computacional, como decomposição de problemas, projeção de sistemas, compreensão de comportamento humano e recursividade são extremamente úteis no aprendizado em cursos de Tecnologia da Informação.
Dentro do tema de Tecnologia da Informação, uma matéria indispensável é a de Estruturas de Dados. Nela são ensinados conceitos como pilhas, filas, listas, algoritmos de busca, complexidade de algoritmos, árvores, *hashing*, entre outros, que continuam a ser utilizados em matérias subsequentes.
Uma ferramenta que tem sido utilizada há um tempo, tem mostrado eficácia no auxílio da aprendizagem de temas relacionados à computação e mais especificamente relacionados à matéria de Estruturas de dados é o uso de *Games*.
Partindo desse pressuposto, o *game* desenvolvido tem o objetivo de justamente, ser uma ferramenta auxiliar no ensino das técnicas e conceitos relacionados ao ensino do tema de Estruturas de Dados.
Após utilização em sala de aula do jogo, foram obtidos resultados positivos quanto ao seu impacto no ensino de algoritmos de ordenação na disciplina de Estruturas de Dados.
Palavras-chave: Pensamento Computacional, Estruturas de Dados, *Games* Educativos, Interfaces Digitais, Jogo

ABSTRACT

In an increasingly digital world, with the rising importance of electronic devices in our daily lives, it is necessary to train professionals in the area of Information Technology, to perform maintenance and innovation functions related to these devices. One of the ways to obtain this training is by joining the University, when entering a Higher Education course.

However, this adhesion does not always occur smoothly, and the subject of Computational Thinking, is a subject that, from my own experience, has its intricacies and nuances that make this learning difficult. Some pillars of Computational Thinking, such as problem decomposition, system design, understanding human behavior, and recursion are extremely useful in learning in Information Technology courses.

Within the subject of Information Technology, an indispensable subject is Data Structures. It teaches concepts such as stacks, queues, lists, search algorithms, algorithm complexity, trees, and hashing, among others, which continue to be used in subsequent subjects.

One tool that has been used for some time, and has shown to be effective in helping the learning of topics related to computing and more specifically related to the subject of Data Structures, is the use of Games.

Based on this assumption, the developed game aims to be an auxiliary tool in the teaching of techniques and concepts related to the teaching of the subject of Data Structures.

After utilising the game in a classroom environment, positive results were obtained as related to its impact in the process of teaching sorting algorithms in the Data Structures course.

Keywords: Computational Thinking, Data Structures, Educational Games, Digital Interfaces, Game

LISTA DE ILUSTRAÇÕES

SUMÁRIO

1 **INTRODUÇÃO**

Com o crescente avanço tecnológico e a necessidade de que todos tenham domínio pelo menos básico de conceitos de computação, não há porque não incorporar essa tecnologia no ensino desses conceitos. Os métodos mais tradicionais de aprendizagem em sala de aula, embora sejam eficientes para demonstrar conceitos abstratos e informações factuais para um grande número de alunos, não é o mais adequado quando o objetivo é obter níveis mais superiores de aprendizagem, que promovam a aplicação dos conhecimentos adquiridos em situações práticas (Choi, J. e Hannafin, M. 1995).

Uma abordagem que apresenta potencial é o uso de Games e Gamificação aplicados à Educação (conforme trabalhos correlatos descritos na seção 3). Por meio de jogos visualmente atrativos e estimulantes, acompanhado de explicações e demonstrações exclusivos a essa mídia, o ensino de cursos relacionados à computação tem a tendência de se tornar mais acessível. Um exemplo que se alinha com essa noção é o trabalho descrito por Dicheva, Irwin e Dichev (2019). Nele, os autores relatam sobre uma pesquisa com um grupo de alunos utilizando uma plataforma denominada OneUp para estudo de Estruturas de Dados. Os resultados obtidos por eles são bem encorajadores, sendo que foi observado um aumento tanto no engajamento dos estudantes em relação à matéria, quanto um aumento significante nas notas dos alunos, principalmente a ocorrências de notas suficientes para passar na disciplina. Além disso, o sentimento geral dos alunos segundo as pesquisas foi de que a plataforma teve um impacto positivo no processo de aprendizagem e aumentou a motivação dos alunos. Outro trabalho que se alinha com esses resultados é o de Daghestani e Ibrahim (2018), que obtiveram resultados parecidos quanto ao aumento de notas e engajamento dos estudantes.

Assim como as tecnologias se desenvolvem com o passar do tempo, o ensino pode também se desenvolver, usufruindo dessas novas técnicas derivadas do avanço tecnológico para a facilitação do aprendizado de conceitos e técnicas relacionadas aos cursos de Tecnologia da Informação e Computação. Como é observado por Okada e Sheehy (2020), grande parte dos alunos tem a percepção de que seria proveitoso ter diversão associada ao processo de aprendizagem, apesar de alguns alunos acharem a diversão no ensino desnecessária ou não

esperada. Isso parece confirmar que os estudantes têm preferências por diferentes formas de aprendizagem, sem descartar a possibilidade de existirem problemas e dificuldades na implementação dos jogos educativos e sua incorporação no processo de aprendizagem. Interessante também é o trabalho (BATISTELLA e VON WANGENHEIM, 2016), onde os autores fazem uma revisão sistemática dos jogos educacionais relacionados ao ensino em computação da época, chegando à conclusão que a maioria dos *games* da época pareciam não ter os fundamentos necessários de jogos convencionais tais como competição, interação e divertimento, talvez diminuindo sua eficácia.

Ainda segundo Daghestani e Ibrahim (2018), supõe-se que a adoção de jogos educativos trazem um aspecto emocional e motivacional perdido na ausência de um professor presencial aos discentes do conteúdo apresentado e que, esses jogos, possivelmente ajudem no ensino a distância e ofereçam algum benefício na transição cada vez maior de cursos inteiramente presenciais para cursos semipresenciais ou inteiramente à distância, visto a atual situação pandêmica mundial e as medidas necessárias para prevenção da doença.

De maneira geral, no ensino de computação existe um equilíbrio entre aulas práticas e teóricas, sendo essa coexistência, a base que fará com que o aluno se adapte, com visão crítica, às novas situações de sua área de formação (MEC[1], 2012).

A disciplina de Estruturas de Dados, sendo em parte uma transição entre as disciplinas mais iniciais e as disciplinas mais avançadas dos cursos onde participa do currículo, tem esse papel de mostrar as ferramentas aprendidas sendo utilizadas em situações mais próximas a suas aplicações em mundo real (CHINN et al, 2003).

Além disso, considerando que as estruturas de dados em si, conceito que dá nome à disciplina, são fatores importantes no momento de arquitetura dos sistemas, principalmente quando se trata do desempenho dessas aplicações (CHINN et al, 2003), pode-se afirmar que um profissional que sabe utilizá-las corretamente possui um diferencial positivo em relação a outros profissionais que, por algum motivo, não tiveram a oportunidade de obter o mesmo conhecimento.

O pensamento computacional, compreendido como a análise e resolução de problemas por meio da construção de algoritmos (MARTINS e ELOY, 2019) e

1 Ministério da Educação

discutido com maior profundidade na subseção 2.2 do presente trabalho, vem ao encontro a essa necessidade de fornecer ao aluno situações mais reais de aplicação dos conceitos aprendidos. Ele contém princípios como reconhecimento de padrões, decomposição de problemas e abstração que ajudam a fazer essa transição de problemas com propósito mais didático a soluções mais práticas.

1.1 – OBJETIVO GERAL

O objetivo do presente trabalho é a elaboração de um jogo educativo, que possa facilitar o aprendizado de conceitos relacionados a algoritmos de ordenação na disciplina de Estruturas de Dados, sendo uma ferramenta disponível para docentes de múltiplos níveis de ensino para esse fim.

O trabalho tem como público-alvo alunos do ensino Superior em caráter introdutório aos temas abordados. Será desenvolvido como um jogo que reforça aspectos de Pensamento Computacional para auxiliar no aprendizado de conceitos computacionais, com um foco maior em Estruturas de Dados.

1.2 – OBJETIVOS ESPECÍFICOS

O1. Sintetizar a fundamentação básica de conceitos como Pensamento Computacional e Jogos Educacionais, além de um estudo sobre Estruturas de Dados;

O2. Procurar exemplos de jogos educativos que tratam de assuntos similares, verificando métodos e padrões de projeto em jogos educacionais similares já existentes;

Q3. Encontrar uma tecnologia para realizar o desenvolvimento do jogo (qual linguagem, se será *Web*, aplicação *Desktop*, etc.);

O4. Desenvolver o jogo educacional utilizando o conhecimento adquirido nos objetivos anteriores;

O5. Avaliar o impacto do jogo desenvolvido por meio de métricas;

2 **FUNDAMENTAÇÃO TEÓRICA**

Nesse capítulo, são abordados temas relevantes que sustentam o projeto. Primeiramente é apresentado o tema de Pensamento Computacional, seguido pelo tema de Estruturas de Dados e finalmente Jogos Educacionais.

2.1 PENSAMENTO COMPUTACIONAL

O pensamento computacional é compreendido como a análise e resolução de problemas por meio da construção de algoritmos (MARTINS e ELOY, 2019). Ele não é necessariamente restrito ao meio de Tecnologias de Informação, já que esses algoritmos, sendo apenas uma palavra para uma sequência de passos para executar determinada ação, podem não ser algoritmos computacionais.

O termo foi cunhado por Wing (2006), que, mesmo naquela época, ressaltava a importância do pensamento computacional como habilidade para não só profissionais da área de Tecnologia da Informação, mas também para as pessoas em geral. Há um foco em:

- Resolução de problemas
- Reconhecimento de padrões
- Decomposição de problemas
- Projeção de sistemas
- Compreensão de comportamento humano
- Recursividade

São feitas uma série de questões como “O quão difícil é resolver esse problema?” e “Qual a melhor forma de resolvê-lo?”, encontrando as respostas em técnicas que não são exclusivas mas que também incluem temas relacionados à Tecnologia da Informação como abstração e decomposição.

Uma frase interessante do autor Paulo Blikstein é a seguinte: “(...) não dá pra redesenhar uma linha de produção, ou decodificar o DNA, copiando e colando textos da internet.”, frase essa que, junto ao resto de sua publicação (BLIKSTEIN, 2008), significa que o Pensamento Computacional tem papel fundamental na criação de novo conhecimento no momento digital em que a sociedade se encontra atualmente. Segundo Blikstein, é necessário além de identificar as tarefas cognitivas que podem ser feitas de forma mais rápida e eficiente por um computador, também ter a

habilidade de programar um computador para realizar essas tarefas previamente identificadas de maneira muito mais rápida do que um ser humano.

Ainda no âmbito da importância do ensino de Pensamento Computacional, a autora Jeanette M. Wing afirma em uma apresentação que realizou (WING, 2008), na seção que contém suas expectativas para o futuro, que o tema será uma habilidade fundamental utilizada por todos no mundo até a metade do século 21, dizendo também que será algo como ler, escrever e aritmética. A autora também provê alguns exemplos de Pensamento Computacional sendo utilizado em outras áreas de pesquisa, como Biologia, Neurociência, Química entre outros.

2.2 ESTRUTURAS DE DADOS

A disciplina de Estruturas de Dados é parte fundamental do aprendizado de Ciência de Computação e cursos relacionados, como Sistemas de Informação e Engenharia de Computação, sendo parte da categoria de Fundamentos de Desenvolvimento de Software (CC2020 TASK FORCE, 2020). A disciplina tem como objetivo geral a compreensão e construção, com o ponto de vista de Orientação a Objetos, das estruturas de dados clássicas, como listas, filas, pilhas e árvores. Além disso, são apresentados os algoritmos de ordenação e busca relacionados a essas estruturas, assim como os ganhos/perdas de desempenho do uso de certas estruturas/algoritmos comparados com os demais (DE LUCCA, 2017).

A disciplina não somente serve de transição de solução de problemas e programação mais básica para tópicos mais avançados em Ciência da Computação, como também é a última disciplina de programação para muitos estudantes (de outros cursos como Engenharia de Computadores e Engenharia Elétrica) (CHINN et al, 2003). Há um foco em situações reais e conhecimento social acumulado pelos estudantes (CHINN et al, 2003) que torna ainda mais importante, em Estruturas de Dados, o emprego das habilidades desenvolvidas pelo Pensamento Computacional previamente citadas, como abstração e pensamento algorítmico, por exemplo.

Quanto ao conteúdo programático da disciplina, inicialmente são apresentados Conceitos de Estruturas de dados (4 horas-aula), seguido por Estruturas Lineares (Pilhas, Filas, Listas – 20 horas-aula), Tabelas de Espalhamento (*hashing* – 22 hora-aula), Árvores (24 horas-aula), Grafos (8 horas-aula), Métodos clássicos de

Ordenação (8 horas-aula) e Organização de Arquivos (8 horas-aula) (DE LUCCA, 2017).

Pode-se observar que o conteúdo foco do jogo desenvolvido, Métodos de Ordenação, é ensinado bem ao fim do conteúdo programático, após o entendimento das Estruturas de Dados mais comuns, sendo um dos assuntos mais complexos aprendidos na disciplina, assumindo que os conteúdos são apresentados de forma que a complexidade aumenta gradativamente durante seu andamento.

Para a facilitação da aprendizagem desses algoritmos de ordenação foi criado o *Sort it Out*. Inicialmente foram implementadas fases apenas com os algoritmos *Bubble Sort* e *Merge Sort*.

No algoritmo *Bubble Sort*, são comparados pares de elementos, fazendo a troca de posição entre eles, mantendo o menor elemento no começo da lista a cada comparação e retornando ao começo da lista se ao fim dos elementos a estrutura ainda não está ordenada.

Já o algoritmo *Merge Sort*, faz uso do método de “divisão e conquista”, onde o algoritmo divide a listagem em 2 sequências de comprimento n/2, ordenando recursivamente cada sequência (dividindo novamente quando possível) e fazendo o merge das duas sequências ordenadas para obter a listagem ordenada completa.

2.3 JOGOS EDUCACIONAIS

Na presente seção, são relacionadas algumas definições do que é um jogo segundo diversos autores. Além disso, são apresentadas algumas classificações de jogos de acordo com certos parâmetros, como qual seu relacionamento com o ensino e qual seu objetivo final (se é puramente lúdico ou tem como objetivo a aprendizagem de algum conceito). Finalmente, são apresentados exemplos de maneiras de utilizar jogos no contexto de aprendizagem.

2.3.1 DEFINIÇÕES DE JOGO

Para o autor Huizinga, em 1930, a definição é mais liberal e se refere mais ao jogo como uma atividade lúdica, voluntária e que traz uma evasão da vida real ao jogador (HUIZINGA, 2003).

A definição de Crawford (1982), se baseia em quatro elementos (CRAWFORD, 1982):

- representação (o jogo apresenta uma representação simplificada e subjetiva da realidade);
- interação (o jogador é capaz de provocar alterações e verificar suas consequências na realidade apresentada);
- conflito (o conflito surge naturalmente a partir da interação do jogador com o jogo e o enfrentamento de seus obstáculos);
- segurança (o ambiente do jogo protege o jogador das consequências que a realidade por ele representada infringiriam ao jogador em um ambiente real, por exemplo um jogo de simulação de direção de automóveis protege o jogador das consequências de bater o carro ou de sofrer um acidente);

Em seu artigo chamado “Os Jogos e os Homens” publicado em 1990, o autor português Caillois definiu como jogo, uma atividade:

- Livre: deve ser voluntária, em virtude de se manter a diversão;
- Delimitada: deve ter limites de espaço e tempo;
- Incerta: desfecho do jogo não deve ser pré-definido;
- Improdutiva: não deve gerar bens, nem riquezas nem elementos novos de espécie alguma;
- Regulamentada: deve ter regras definidas;
- Fictícia: deve estar separada das atividades e consequências do mundo real;

É interessante observar o contraste entre as ideias de Caillois e Huizinga, sendo que Caillois publicou sua obra depois de Huizinga, e, apesar de destacar pontos positivos trazidos pela obra do outro autor, pensa que o jogo é um produto da cultura da sociedade e não o contrário como Huizinga afirma (PICCOLO, 2008).

Existe a definição de Juul (JUUL, 2005), onde há seis requisitos a serem satisfeitos pelo jogo:

(1) ser um sistema formal baseado em regras,

(2) ter resultados variáveis e quantificáveis,

(3) a cada resultado deve ser possível associar valores distintos,

(4) os jogadores devem se esforçar para influenciar o resultado do jogo,

(5) os jogadores devem se sentir emocionalmente ligados aos resultados e

(6) as consequências de sua atividade devem ser opcionais e negociáveis.

Mais atualmente, para Becker, um jogo deveria ter as seguintes propriedades (BECKER, 2021):

- Interativo;
- Tenha regras;
- Tenha um ou mais objetivos;
- Tenha uma medida quantificável de progresso (ou sucesso);
- Tenha um fim reconhecível;

2.3.2 CLASSIFICAÇÕES DE JOGOS EDUCACIONAIS

Jogos sérios (Serious Games) são um subconjunto de jogos. Eles são desenvolvidos com propósitos que não são ou que apenas incluem entretenimento. Incluem jogos de saúde (de treinamento cirúrgico ou de diminuição de dor, etc) jogos para mudança (que envolvem problemas de justiça social), *advergames* (jogos “propaganda”, que servem como publicidade), entre outros diversos (BECKER, 2021).

Aprendizagem baseada em jogos (Game-Based Learning) são o processo e a prática de aprender utilizando jogos, do ponto de vista de quem aprende. Nesse caso essa técnica não é um jogo, mas sim uma abordagem de aprendizado, utilizada para melhorar a experiência de aprendizagem (BECKER, 2021).

Pedagogia baseada em jogos (Game-Based Pedagogy), assim como o termo anterior, se trata de uma técnica de aprendizagem que utiliza jogos para melhorar a experiência de aprendizagem, porém nesse caso é observado o ponto de vista de quem ensina, em vez do aluno (BECKER, 2021).

Gamificação (*Gamification*) é o uso de elementos de um jogo (progresso, níveis, ranques, desafios) fora do contexto de jogos. Normalmente é utilizada como forma de motivação, porém também pode ser aplicada para se fazer algo mais próximo a um jogo propriamente dito. A *Gamification* não necessariamente é utilizada para aprendizagem e pode ser utilizada em qualquer contexto (BECKER, 2021).

2.3.3 MANEIRAS DE UTILIZAÇÃO DE JOGOS NO ENSINO DE COMPUTAÇÃO

Os jogos possuem diversas aplicações quando se trata de seu auxílio no aprendizado em Computação. Dependendo da maneira utilizada de apresentação do

jogo educativo, a dinâmica em sala de aula pode mudar significativamente. Alguns desses modos acabam nem utilizando jogos em si, porém utilizam características de jogos para motivar os alunos, ajudando no processo de aprendizagem.

Uma maneira de utilizar jogos na educação é fazer atividades em sala de aula, promovendo interação entre os alunos e tornando o processo de aprendizagem uma atividade competitiva (se o objetivo é ganhar uma partida por exemplo) ou cooperativa. No caso do jogo de cartas *Elementais RPG, o* jogo foi uma atividade aplicada em sala de aula, onde os alunos jogavam partidas do jogo de cartas um contra o outro, com o objetivo de ganhar a partida (OLIVEIRA, et al, 2021). Essa abordagem competitiva tem potencial de ter eficácia em motivar os alunos a se dedicarem à atividade, porém pode gerar experiências negativas para os alunos que são derrotados, principalmente se isso ocorre com frequência durante a atividade. Talvez a abordagem cooperativa seja mais segura, já que incentiva os alunos a trabalhar em equipe e não precisa necessariamente de um time “perdedor” para ser conduzida. Entretanto, sem um objetivo claro, pode ser difícil manter a atenção dos alunos por longos períodos.

Outra abordagem que pode ser observada, é utilizar o jogo não como foco da aprendizagem, mas como ferramenta de facilitação da aprendizagem. Nessa vertente, o jogo é utilizado em conjunto com explicações ou exposições de um professor ou instrutor, que prepara o aluno para aprender um conceito, utilizando depois o jogo para reforçar os conceitos aprendidos anteriormente, Essa é a abordagem utilizada na aplicação *Edubot,* onde a atividade com o jogo é realizada em conjunto com uma aula explicativa antes, para introduzir os conceitos a serem colocados em prática no momento da atividade (LIMA, Thamyla Maria de Sousa et al, 2018). Essa também é a abordagem escolhida para o jogo *Sort it Out*, objeto desse trabalho, visto que ela diminui a necessidade de extensos tutoriais para explicar conceitos dentro do jogo, facilita o desenvolvimento da aplicação e não invalida materiais que já eram utilizados anteriormente na condução da aula, apenas serve como mais uma ferramenta para o professor.

Outra forma de aplicar elementos de jogos no processo de aprendizagem é como o feito por alunos da Universidade Estadual do Norte do Paraná (UNEP) (ARIMOTO, Maurício M. e CRUZ, José Henrique R., 2020), que realizaram oficinas de programação com pontuação, favorecendo os alunos mais dedicados. Nesse

caso não se observa um jogo em si, mas uma *gamificação* do processo de aprendizagem, que incorpora apenas alguns elementos (obstáculos a serem ultrapassados, regras definidas, competição entre jogadores) do que é a definição de um jogo.

2.4 *GAMIFICATION DESIGN CANVAS*

Um artefato utilizado foi uma definição inicial criada sobre o aplicativo, tendo em base a adaptação do Business Canvas Model feita pelo Instituto Tecnológico e de Estudos Superiores de Monterrey (MONTERREY, 2016). O canvas faz parte de materiais didáticos disponíveis no site da instituição e pode ser utilizado em sala de aula para construção de jogos por parte dos alunos. Seu uso também prevê pontuação e premiação (simbólica) para alunos que entregam a atividade pronta. No site da instituição, junto com o *Canvas,* são presentes outros materiais e apresentações sobre Gamificação e seu uso no contexto educativo.

O *Canvas* é separado em oito seções:

- Objetivo: Descreve-se o objetivo principal da criação do jogo.
- Dinâmica: São descritas as regras do jogo de maneira resumida e geral.
- Mecânica: Conceitos com os quais o aluno interagirá para obter progresso no jogo.
- Componentes: Elementos principais utilizados no jogo.
- Estética: Identidade visual e estética do jogo criado.
- Perfil de jogadores: Perfil dos jogadores que são público do jogo criado.
- Gestão (rastreamento e monitoramento): Como será feito o monitoramento do progresso.
- Riscos Potenciais: Riscos que são presentes na criação do jogo.

A seguir, a Figura 1 é uma tradução da versão original em espanhol do *Canvas*.

Figura 1 – *Canvas* de Design de Gamificação

Fonte: Tradução do *Canvas* original em espanhol, que pode ser encontrado na página: https://innovacioneducativa.tec.mx/es/recursos-pedagogicos/estrategias-de-aprendizaje-activo/gamification

3 **TRABALHOS CORRELATOS**

Como trabalhos correlatos foram selecionados jogos que auxiliam no ensino não só de Estruturas de Dados, como também jogos direcionados ao ensino de Pensamento Computacional e outros conceitos de Computação de forma geral, já que foram observados paralelismos em muitas das soluções e problemas encontrados pelos autores. Inicialmente, os termos de pesquisa foram mais gerais como “Jogos Educativos” e “Pensamento Computacional”. Após a escolha de focar mais especificamente o trabalho no desenvolvimento de um jogo educativo para utilização em aulas de Sistemas de Informação foram pesquisados termos mais específicos, como “Jogos Ensino Computação” e “Jogos Ensino Estruturas de

Dados". Foram pesquisados trabalhos nas plataformas Google Scholar[2] e IEEE Xplore[3], além da Biblioteca Digital da Sociedade Brasileira de Computação[4] e Teses de Conclusão de Curso passados da Universidade Federal de Santa Catarina. Foram excluídos trabalhos que não mencionavam algum dos conceitos presentes na fundamentação teórica, como Pensamento Computacional, Educação em Computação ou Estruturas de Dados, por exemplo. São apresentados os trabalhos em duas categorias de acordo com seu relacionamento ou não com a disciplina de Estruturas de Dados, foco do presente trabalho.

3.1 ENSINO DE COMPUTAÇÃO NÃO RELACIONADO A ESTRUTURAS DE DADOS

Um exemplo de jogo utilizado no ensino de técnicas e conceitos de Computação, sem que esse foco seja relacionado à Disciplina de Estruturas de Dados é o jogo Space Code (ROSA; COELHO 2018).

Figura 2 – Jogo Space Code

Fonte: https://repositorio.ufsc.br/handle/123456789/192170; Imagem na página 38 do PDF

2 https://scholar.google.com/
3 https://ieeexplore.ieee.org/
4 https://sol.sbc.org.br/

O jogo consiste na movimentação de um foguete por um tabuleiro, sem esbarrar nas paredes ou obstáculos no caminho, capturando todas as estrelas existentes no cenário. Para que a nave se movimente, são utilizadas peças físicas também elaboradas pelos autores, que simbolizam instruções aceitas pelo jogo e, sendo organizadas na ordem correta, formam uma sequência lógica que leva o foguete até seu destino final. Há dois conjuntos de peças disponíveis. Um contém símbolos que representam uma estrela, um círculo e um triângulo, além de setas para a direita e para a esquerda e um símbolo de seguir em frente. Já o segundo conjunto contém peças que são utilizadas para montar uma estrutura de repetição.

Organizadas as peças conforme a lógica do jogador, o mesmo deve tirar capturar uma imagem utilizando a câmera de seu dispositivo móvel, para que, sendo processada essa imagem, as peças sejam identificadas pelo aplicativo. Após esse passo, o aplicativo executa as instruções desejadas, mostrando à criança a movimentação do foguete pelo tabuleiro de acordo com as instruções previamente definidas.

No trabalho “Uma Proposta de Jogo em Duas Etapas para Conhecer a Computação”, é feito o uso de um jogo em duas etapas como forma de apresentar algumas disciplinas de cursos de graduação em Computação para estudantes que estão prestes a ingressar no ensino superior (SENA, Alexandre da Costa et al., 2019).

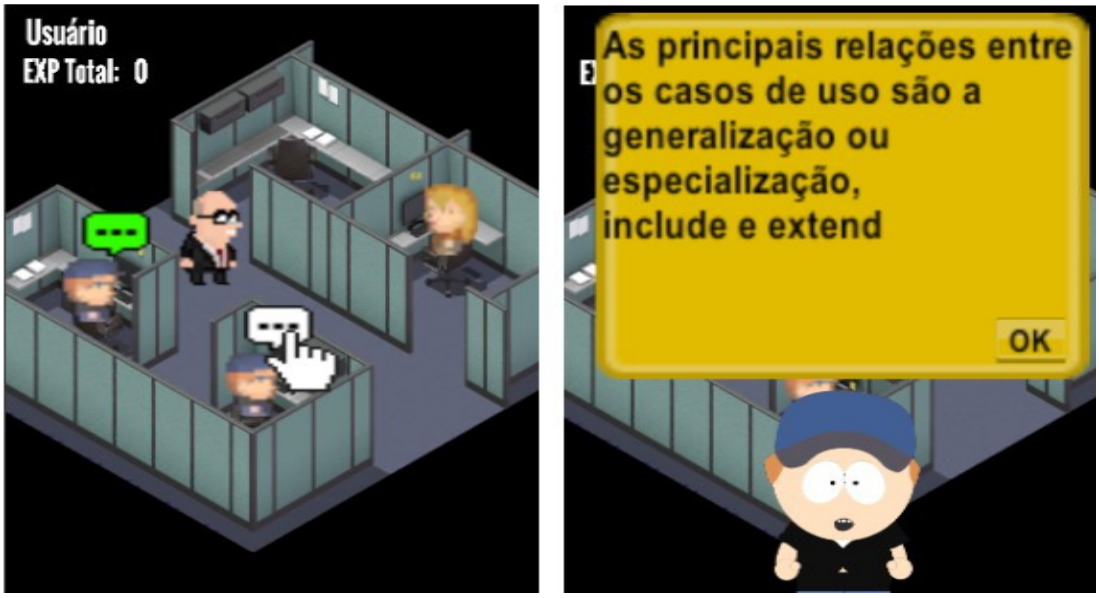
O foco é apresentar a estudantes egressos do ensino médio conceitos relevantes aos cursos da área de computação do ensino superior. Disciplinas como Algoritmos, Redes de Computadores, Arquiteturas de Computadores e Otimização em Grafos são abordadas pelos autores.

Na primeira etapa do jogo proposto são feitas apresentações sobre os conceitos previamente mencionados. Na segunda etapa, é utilizado um de quatro minijogos feitos em Unity3D com a linguagem C#, que correspondem a um dos temas das disciplinas escolhidas. Cada minijogo associa o conteúdo ensinado com o jogo, explica o conceito de maneira formal e mostra aplicações na vida real do conceito.

Outro trabalho relevante é o jogo *Retraining*, uma solução computacional para apoiar o ensino de especificação de requisitos (DAMASCO, 2021). Possui foco maior em Engenharia de Software, processo de compreensão e definição dos

serviços requisitados do sistema e identificação de restrições relativas à operação e ao desenvolvimento do sistema (SOMMERVILLE, 2011).

Figura 3 – Tela do jogo Retraining



Fonte: https://repositorio.ufsc.br/handle/123456789/228407; Imagem na página 64 do PDF

O jogo utiliza de aprendizagem ativa para, através de um *RPG*, o aluno entenda e ponha em prática os conceitos da matéria anteriormente mencionada. No jogo o aluno assume o papel de um estagiário em um escritório, que além de cumprir funções tipicamente associadas com estagiários como buscar café, documentos, entre outros, também participa de projetos e passa por treinamentos. Após o treinamento que consiste em perguntas de conhecimento geral após receber informações sobre um determinado conteúdo, o aluno, se acertar uma certa quantidade de perguntas passa para a fase seguinte. Na fase seguinte o aluno colabora com a equipe em projetos. No final é disponibilizado um *Ranking.*

Foi realizada a avaliação do jogo utilizando-se o modelo MEEGA+[5], obtendo-se resultados positivos apesar do número pequeno de amostras.

Outra abordagem é a proposta por um grupo de alunos da Universidade Federal de Pelotas (RS) (OLIVEIRA, 2021), que criaram o jogo de tabuleiro

5 O MEEGA (Model for the Evaluation of Educational Games) é um modelo que tem como objetivo a avaliação da qualidade e dos reais benefícios trazidos por jogos educativos como recursos didáticos, principalmente quando se trata de seu uso no processo de ensino de engenharia de software, além da avaliação de sua aplicabilidade, utilidade, validade e confiabilidade (SAVI, 2011). MEEGA+ é uma extensão do modelo MEEGA feita por parte dos autores do modelo original junto a novos co-autores (PETRI et al., 2019).

Elementais RPG, baseado no gênero *RPG*[6] de jogos sérios, que consiste de personagens relacionados a um certo elemento (fogo, água ou planta) que possuem vantagens ou desvantagens entre eles mesmos, baseando-se em que elemento eles possuem (água ganha de fogo, planta ganha de água, fogo ganha de planta, por exemplo). Dessa forma, os participantes do jogo (crianças do quarto ano do ensino fundamental), são incentivados a pensar de maneira lógica e chegar à conclusão de que elementos tem vantagem contra que elementos. Essa proposta se destaca em relação às anteriores que foram exploradas no presente trabalho, pois se trata de um jogo não digital, o que reforça a ideia de que se pode ensinar conceitos relevantes ao contexto de Pensamento Computacional sem que seja feito o uso de dispositivos digitais.

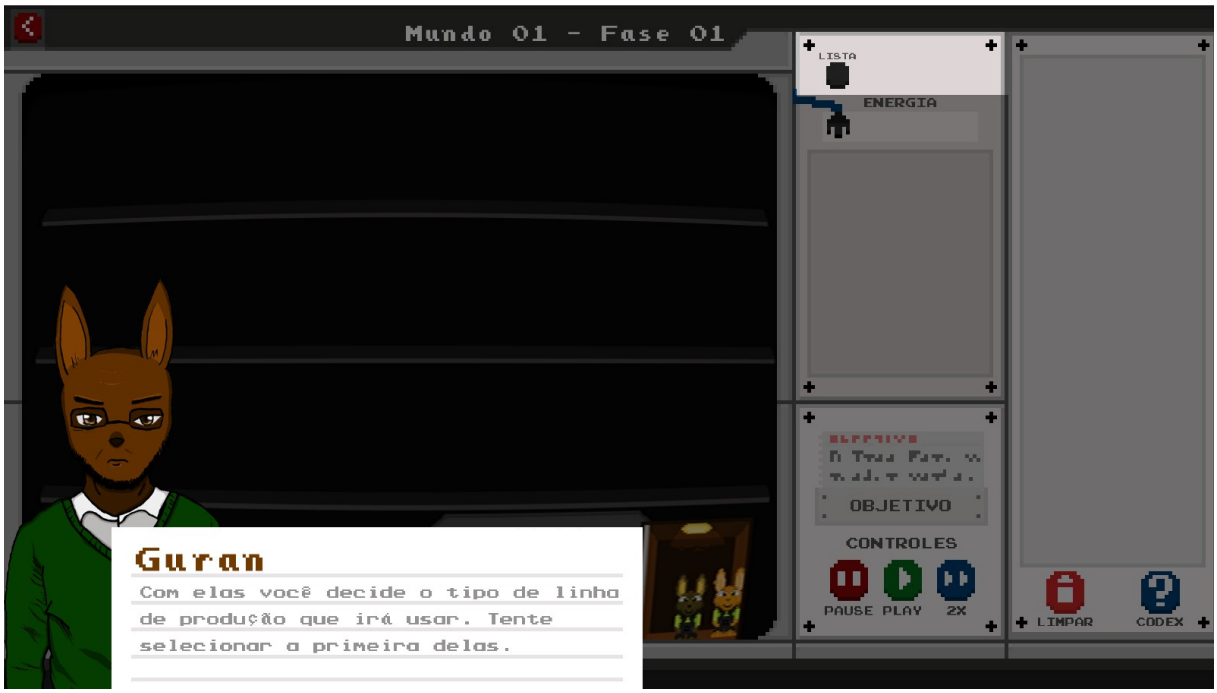
3.2 ENSINO DE TÉCNICAS E CONCEITOS DE COMPUTAÇÃO COM FOCO EM ESTRUTURAS DE DADOS

Na presente categoria, são apresentados alguns jogos que foram criados para auxiliar no ensino de conceitos de computação com foco maior na disciplina de Estruturas de Dados.

O primeiro trabalho correlato dessa categoria é o Iron Ears (DA ROSA et al, 2020), desenvolvido por alunos da UFPR (Universidade Federal do Paraná), tendo foco o Ensino Superior. O jogo é de quebra-cabeças, 2D, disponível em inglês e português.

6 Role-playing game: Em português “Jogo de Interpretação de Papéis”, fortemente influenciados por literatura fantástica, geralmente envolvem personagens controlados cada um por um jogador, onde cada jogador escolhe as habilidades e ações de seu personagem através de uma campanha pré-feita (VEUGEN, 2004).

Figura 4 – Tela de tutorial do jogo Iron Ears



Fonte: Screenshot criado do próprio jogo em 07/03/2022. Versão 0.9.3 disponível em: https://npc42-games.itch.io/ironears

Foi criado com a utilização da engine Unity[7]. Destaca-se a utilização de um enredo bastante elaborado, com personagens antropomórficos.

Quanto à jogabilidade do Iron Ears, o jogador é desafiado a ajudar na gestão de uma linha de montagem de produção de robôs. Utilizando comandos disponíveis em um menu lateral, por meio de *drag-and-drop*, o jogador controla as ações das máquinas e personagens na linha de montagem. Contém uma grande quantidade de documentação *in-game*, que pode auxiliar o jogador tanto a entender melhor os conceitos ensinados quanto a entender melhor que atividades deve desenvolver no jogo. O jogo foi então aplicado em sala de aula na disciplina de Estruturas de Dados I na UFPR, onde obteve resultados positivos no processo de ensino-aprendizagem.

Outro trabalho interessante é o *The Stack Game* ou em português “O Jogo de Pilha”, referenciando a estrutura de dados chamada pilha (DICHEVA, Darina; HODGE, Austin; 2018). O jogo tem o objetivo de ensinar o conceito abstrato de pilha (relacionamento lógico entre seus elementos e as operações que podem ser

7 Unity é uma plataforma de criação e Ambiente Integrado de Desenvolvimento, para criar mídia interativa, típicamente jogos. A sua primeira versão (1.0.0) foi criada pelos colegas: David Helgason, Joachim Ante and Nichoals Francis na Dinamarca. (HAAS, John K., 2014)

realizadas na estrutura), como implementar a estrutura em alguma linguagem de programação e como utilizar pilhas para resolver problemas.

Figura 5 – Figura da parte 1 level 2 do jogo The Stack Game

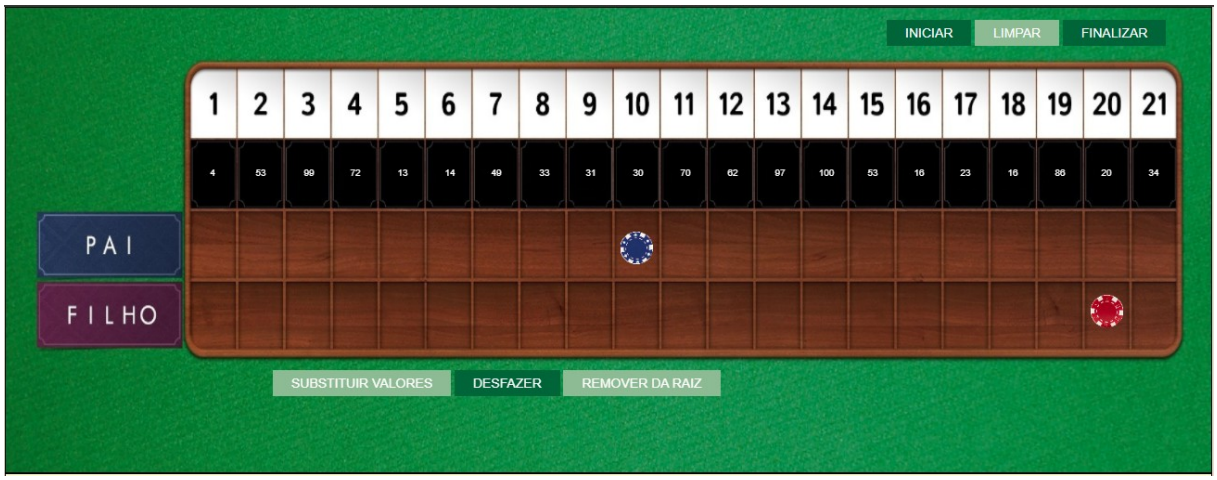
Fonte: https://www.researchgate.net/figure/The-Stack-Game-Part-1_fig1_311607952

No jogo, o jogador controla um robô que está na viajem de volta para casa após completar uma missão. A nave espacial do robô, no entanto cai e ele acaba tendo que caminhar, enquanto passa por vários obstáculos/portas, que devem ser desbloqueados através da resolução de problemas relacionados ao funcionamento de pilhas. Cada parte do jogo inclui vários níveis com dificuldade incremental das tarefas. O *The Stack Game* foi utilizado junto a explicações em aula dos conceitos aplicados, sendo utilizado como ferramenta de auxílio na aprendizagem.

Após os devidos testes, os resultados obtidos pelos autores mostram não somente que o jogo foi efetivo quando se trata das notas observadas nas provas conduzidas, quanto o feedback dos alunos em relação ao jogo foi majoritariamente positivo.

Um trabalho interessante é o jogo SORTIA 2.0, focado no ensino de um algoritmo de busca chamado *Heapsort* através da simulação da sua execução (BATTISTELLA, Paulo E. et al., 2016). O jogo é jogado por um jogador apenas, pode ser jogado *online* e é gratuito. Um destaque do trabalho é a utilização do modelo MEEGA de avaliação de jogos educativos para fazer o estudo de caso do jogo desenvolvido.

Figura 6 – Jogo Sortia 2.0



Fonte: https://incod.ufsc.br/sortia/

SORTIA 2.0 se trata da segunda versão do jogo SORTIA, desenvolvido previamente por três dos autores da segunda versão, tendo sua primeira versão publicada como trabalho no 23° Simpósio Brasileiro de Informática na Educação (Battistella, P. E., Wangenheim, A. von e Wangenheim, C. G. Von, 2012). Em relação à primeira versão o jogo apresenta uma interface aprimorada e um maior nível de atratividade, problemas observados pelos autores no desenvolvimento da primeira versão. Após o desenvolvimento do jogo, ele foi aplicado em sala de aula na disciplina de Estruturas de Dados do curso de Ciência da Computação da UFSC, no segundo semestre de 2015. Antes que os alunos jogassem o jogo, foi feita uma aula expositória pelo professor da disciplina, explicando o algoritmo *Heapsort.* Segundo a avaliação dos resultados obtidos após a aplicação do modelo MEEGA, os alunos consideraram que o jogo contribuiu para seu aprendizado, que foi mais eficiente que outras atividades da disciplina e houve um aumento significativo da fixação do conteúdo por parte dos alunos.

Um exemplo de jogo sério (*Serious Game) DS-Hacker* (ROJAS-SALAZAR, RAMÍREZ-ALFARO e HAAHR; 2020), um jogo de ação/aventura *3D*, feito em *Unity*, onde o usuário assume o papel de um *hacker* robótico que deve percorrer um labirinto do qual ele deve extrair informações. O jogo tem como foco o ensino de Estruturas de Dados, mais especificamente da estrutura e das propriedades da busca binária em árvores. Em seu artigo, os autores relatam os resultados de um estudo piloto, utilizando o jogo por eles desenvolvido após duas situações diferentes: (1) fazendo uso do *DS-Hacker* e (2) lendo um material escrito e assistindo dois

vídeos. Após os testes foram mostrados os resultados de um ponto de vista de usabilidade do jogo, satisfação do usuário e a percepção dos participantes dos métodos utilizados pelo jogo para transmitir o conceito de Busca Binária em Árvores apresentado.

Os resultados demonstraram que os participantes conseguiram relacionar os conceitos apresentados com suas contrapartes no jogo, além de terem a sensação que aprenderam com o jogo e de que gostaram de forma geral da experiência. Embora o grupo que utilizou o jogo não tenha “ganhado” do grupo de controle em todas as etapas do teste, ele apresentou um progresso mais linear que o outro grupo.

Tabela 1 – Trabalhos correlatos por categorias

	Estruturas de Dados	Digital	Mobile	Puzzle	Para uso em sala de aula	Avaliado
Space Code	Não	Parcial-mente	Sim*	Não	Não	Não
Retraining	Não	Sim	Não	Não	Não	Sim
Elementais *RPG*	Não	Não	Não	Não	Sim	Sim
Iron Ears	Sim	Sim	Não	Sim	Não	Não
The Stack Game	Sim	Sim	Não	Sim	Sim	Não
Sortia 2.0	Sim	Sim	Não	Sim	Sim	Sim (MEEGA+)
DS-Hacker	Sim	Sim	Não	Não	Não	Sim
Sort it Out	**Sim**	**Sim**	**Sim**	**Sim**	**Sim**	**Sim (MEEGA+)**

Fonte: Autoria própria

* O jogo *Space Code* tem uma parte que é desenvolvida em Mobile, porém como o jogo não é completamente digital, sua categorização como jogo Mobile é discutível.

4 PROPOSTA

Tendo em mente a importância dos conceitos de Pensamento Computacional, mencionados na seção 2.1 do presente trabalho, é proposta a criação de um jogo digital, em forma de aplicativo mobile, a fim de auxiliar o processo de abstração e a visualização dos conceitos de algoritmos de busca ensinados na disciplina de Estruturas de Dados.

Tomando proveito dos benefícios observados por outros autores, também discutidos anteriormente na seção de trabalhos correlatos (seção 3), procura-se fomentar competição e aumentar a motivação dos alunos, impactando de forma positiva o processo de aprendizagem dos alunos.

A partir do estudo dos trabalhos correlatos, optou-se pela utilização de uma interface interativa, por meio de *drag-and-drop*, para permitir que os alunos consigam realizar os passos dos algoritmos de busca, aumentando seu engajamento com os conceitos apresentados.

Além disso, conforme observado na seção 2.3 e suas subseções, que tratam principalmente de aspectos de categorização e utilização de jogos no processo de aprendizagem como ferramenta auxiliar, a ideia inicial é a criação de um jogo que possa tanto ser utilizado em conjunto com aulas expositivas previamente aplicadas, quanto de maneira *standalone* após a aprendizagem dos conceitos, em caráter de revisão.

De forma geral o jogador tem acesso a diferentes algoritmos, que aumentam em complexidade, acompanhando a ordem de ensino observada na disciplina de Estruturas de Dados. Cada algoritmo tem uma fase correspondente, onde o aluno arrasta os elementos de alguma estrutura de dados, de acordo com o algoritmo sendo ensinado, até que complete a execução do algoritmo, tendo como resultado a estrutura de dados ordenada.

Foi feita uma avaliação de que linguagem utilizar, baseando-se em que linguagens o desenvolvedor conhecia, qual era o esforço necessário para aprender novas tecnologias e se esse esforço fazia sentido dado os limites de tempo que são naturais com esse tipo de trabalho. Foram consideradas as tecnologias *Java*, *Javascript*, *Angular* (*Typescript*), *Kotlin* ou o uso de alguma *game engine*.

Como o jogo é do gênero *puzzle* e não contém elementos normalmente associados a *engines* gráficas como personagens, projéteis, cenário, inimigos, foi decidido que o foco seria nas linguagens mais tradicionais para aplicações de uso geral. Além disso, a ideia principal era que o jogo fosse graficamente leve e tivesse compatibilidade com outras plataformas (como Web e iOS). Por limites de tempo o jogo não foi testado na plataforma iOS e Web, porém há a possibilidade futura de melhorar/viabilizar a performance do jogo nelas.

A linguagem escolhida, dadas essas características desejadas foi o *Dart*[8], em conjunto com o *Flutter*[9].

O *Flutter* é uma *framework open-source* de desenvolvimento de aplicações multiplataforma tendo uma só base de código. Isso permite que o desenvolvedor crie uma aplicação que seja executável tanto em iOS quanto em Android ou navegadores Web, sem ser necessário programar usando as linguagens nativas dessas plataformas, porém compilando os aplicativos nessas linguagens nativas.

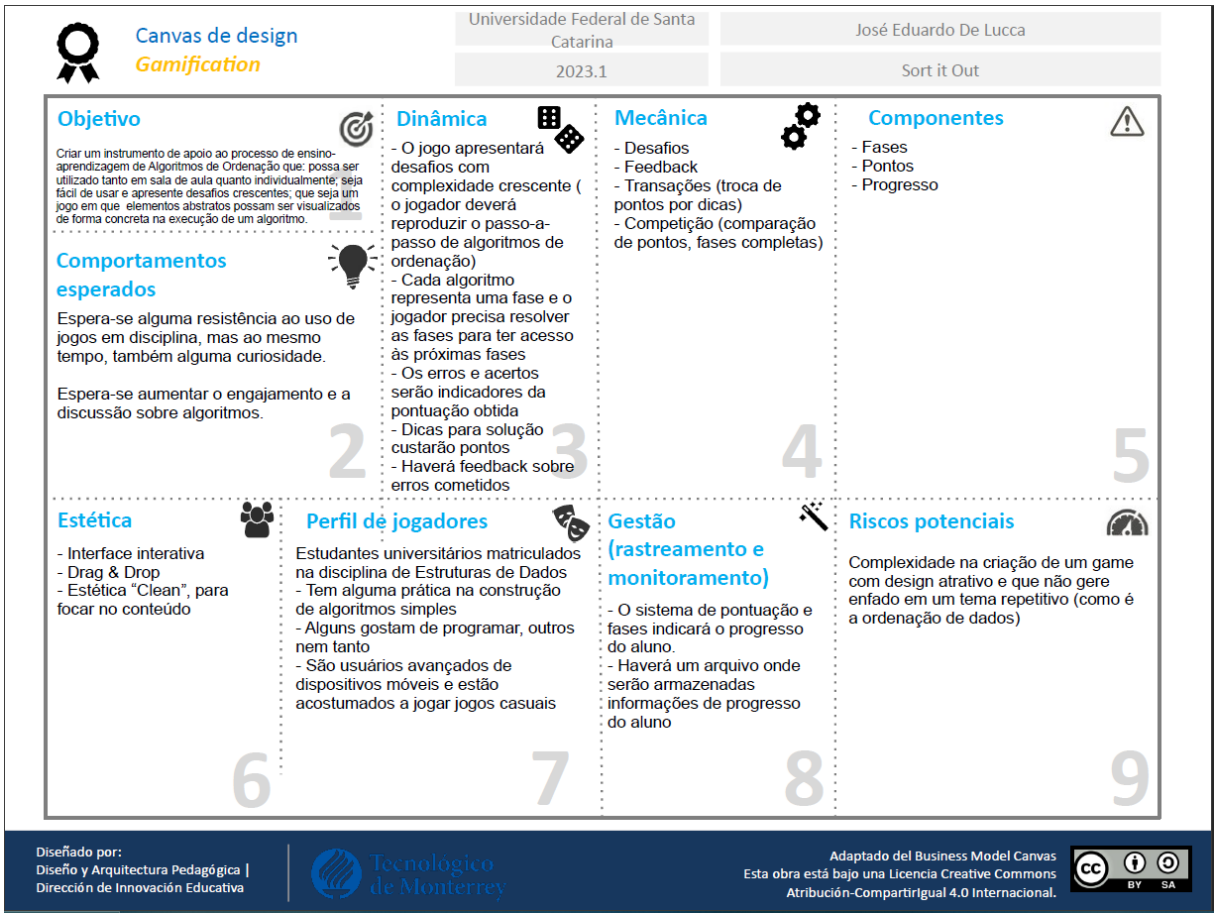
Quanto à sua utilização no desenvolvimento, já que a maioria das estruturas gráficas eram listas reordenáveis e não reordenáveis, com alguns botões simples, o *Flutter* pareceu suficientemente eficiente para a criação do aplicativo.

4.1 *GAMIFICATION DESIGN CANVAS*

Baseando-se na estrutura do *Gamification Design Canvas,* anteriormente apresentado na seção de fundamentação teórica, foi criado um *Canvas* para o jogo *Sort it Out* descrevendo suas características principais e conceitos considerados durante o seu *design.*

8 https://dart.dev/
9 https://flutter.dev/

Figura 7 – Canvas da primeira iteração de definição do aplicativo



Fonte: https://innovacioneducativa.tec.mx/es/recursos-pedagogicos/estrategias-de-aprendizaje-activo/gamification

O *Canvas* é separado em oito seções que serão apresentadas brevemente, assim como seu conteúdo no contexto do jogo *Sort it out.*

– Objetivo: criar um jogo que auxilie no processo de aprendizagem de Algoritmos de Ordenação.

– Dinâmica: terá complexidade crescente, cada algoritmo representa uma fase, os erros e acertos afetam a pontuação, dicas custarão pontos e haverá *Feedback* sobre erros cometidos.

– Mecânica: Desafios, Transações, *Feedback* e Competição.

– Componentes: Fases, Pontuação e Progresso de fases.

– Estética: *design* simplificado e o mais limpo possível.

– Perfil de jogadores: alunos do Ensino Superior, matriculados na disciplina de Estruturas de Dados.

– Gestão (rastreamento e monitoramento): feito por um arquivo que contém as informações de progresso do jogador.
– Riscos Potenciais: que o jogo não tenha *design* atraente o suficiente ou que o jogo seja repetitivo demais para ter um efeito positivo.

4.2 GDD

Antes do início do desenvolvimento do jogo, inspirado pelo trabalho “AVALIAÇÃO DE JOGO PARA APOIO AO PROCESSO DE ENSINO-APRENDIZAGEM DE TÉCNICA DE ESPECIFICAÇÃO DE REQUISITOS DE SOFTWARE”, que faz o *redesign* do jogo Mike Help Me após fazer uma avaliação do jogo (VENÂNCIO, 2022), foi feito um GDD (Game Design Document) para o jogo *Sort it Out*.

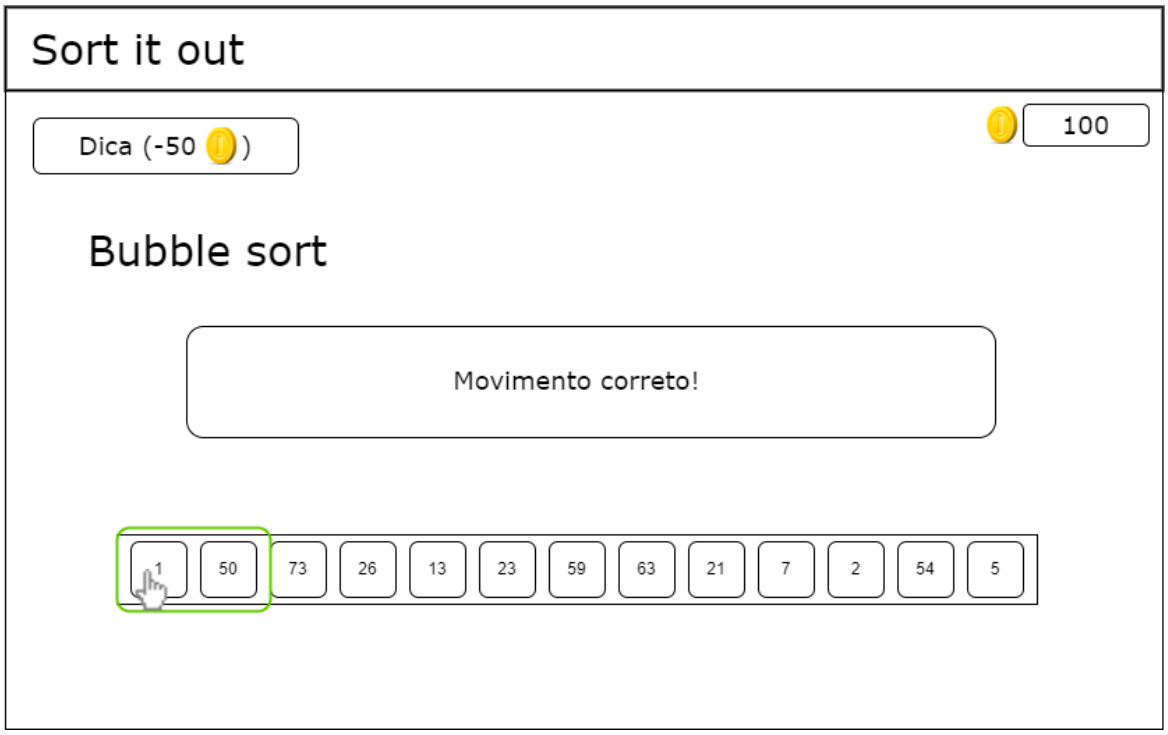
O GDD é um documento para auxiliar no entendimento geral de como o game deverá ser desenvolvido, além de descrever certos aspectos do jogo (SCHUYTEMA apud VENÂNCIO, 2022). O documento é estruturado em tópicos, que descrevem características do jogo, desde seu enredo até sua aparência, sons, e inspirações para seu desenvolvimento.

Tomando como base o GDD desenvolvido para o jogo *Mike Help Me* (VENÂNCIO, 2022), foi feito um GDD para o jogo *Sort it Out*, fazendo ajustes onde foi achado necessário, dado as diferenças de design de ambos os jogos. O GDD se encontra no **Apêndice A** do presente trabalho.

4.3 CONCEPÇÃO DE TELAS

Após a criação dos artefatos apresentados anteriormente, foram criados protótipos básicos das telas que se encontram no aplicativo. Inicialmente apenas foram criados protótipos para as fases que, no aplicativo atual, são do algoritmo *Bubble Sort*.

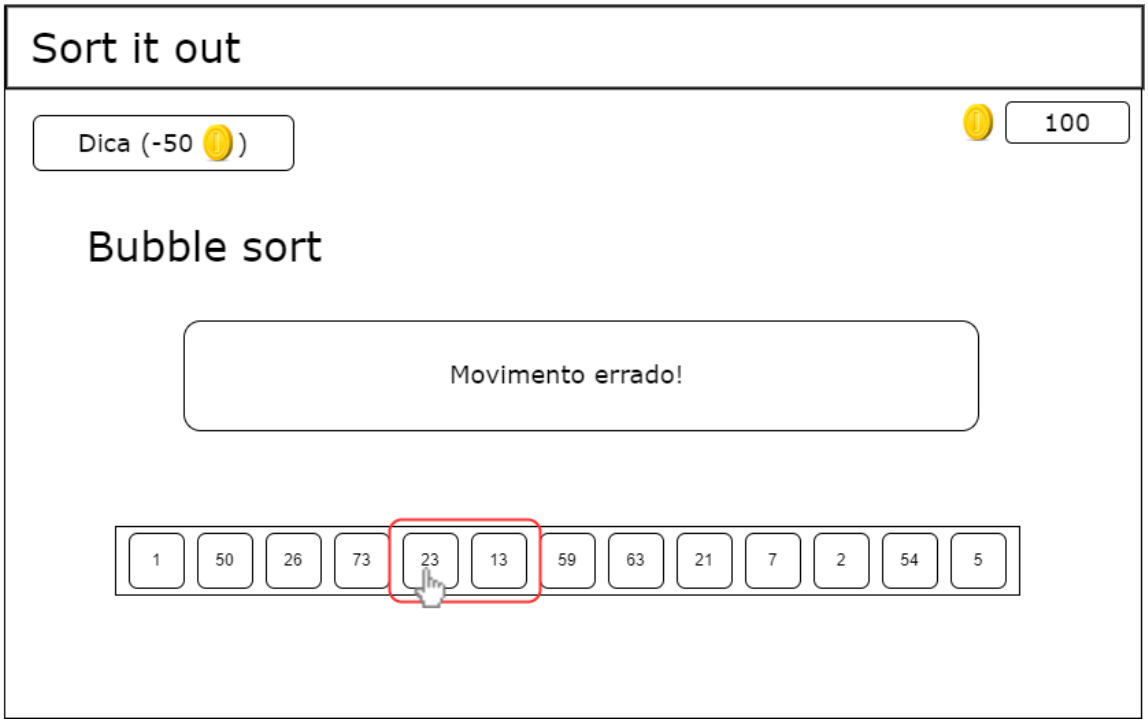
Figura 8 – Esboço tela de movimento correto



Fonte: Autoria própria

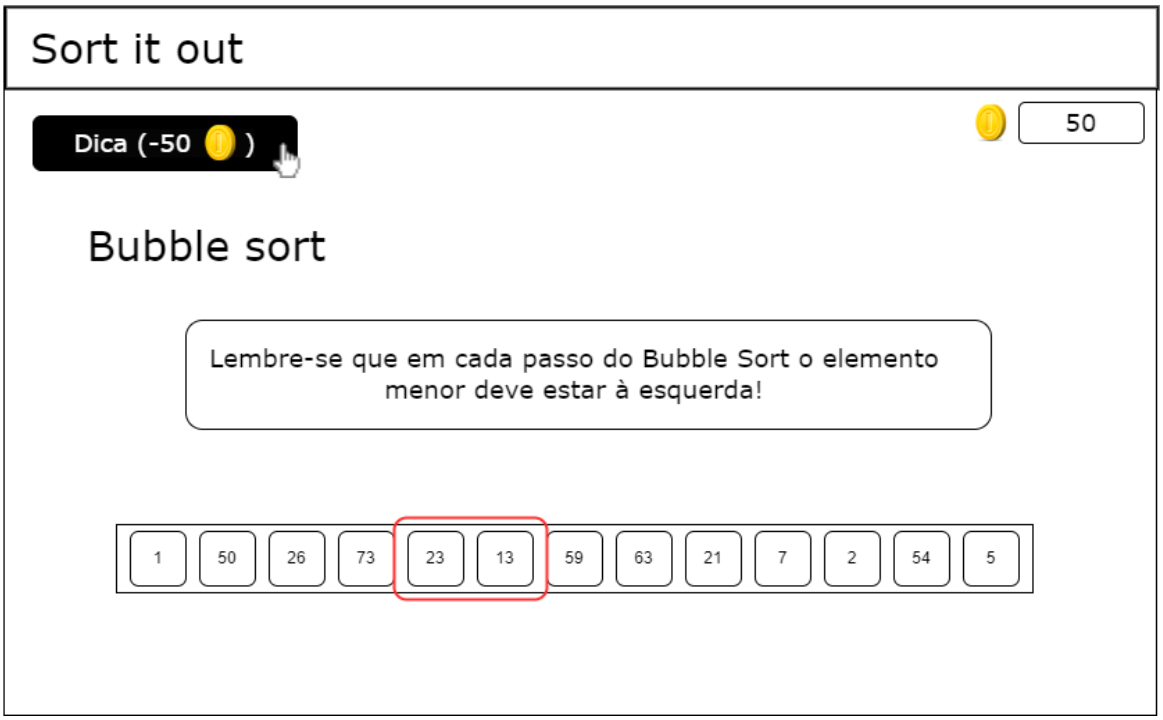
São apresentadas dicas e mensagens de erro, explicando o que o aluno fez de errado. Essas dicas são mostradas até um certo ponto sem “custo” algum ao aluno, mas posteriormente custam moedas ou pontos obtidos apenas por jogar o jogo. Esses pontos são deduzidos do jogador em cada fase, a medida que o jogador comete erros na execução do algoritmo.

Figura 9 – Esboço tela de movimento errado



Fonte: Autoria própria

Figura 10 – Esboço situação onde o jogador solicita uma dica



Fonte: Autoria própria

5 SORT IT OUT

Neste capítulo é apresentado o jogo *Sort it out*, assim nomeado por causa da expressão “*sort it out*” que significa algo como “se vire” ou “resolva”, contendo a palavra “*sort*” que também é utilizada com o sentido de “ordenar” para algoritmos de ordenação (*Sorting Algorithms*).

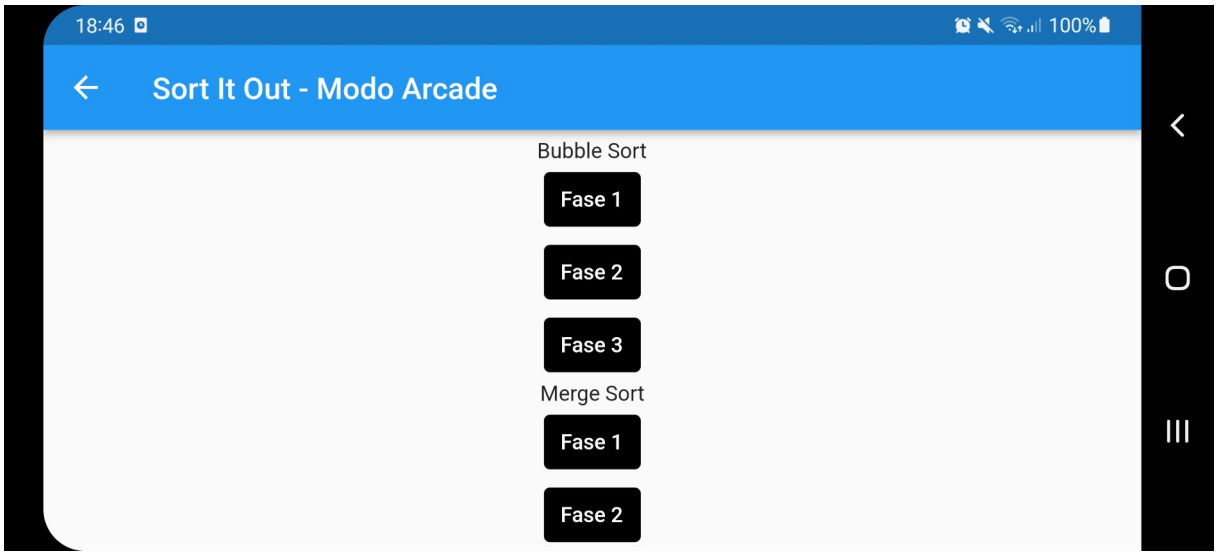
5.1 INTRODUÇÃO

O jogo inicia com uma tela de menu que contém duas opções: Modo Arcade e uma opção de configurações.

Modo Arcade

No jogo as fases são sequenciais, há um fluxo natural de aprendizagem, onde o usuário só consegue jogar uma fase mais avançada após completar os passos anteriores. Os passos são gradativamente mais difíceis, sendo que funções como pedir dicas e mensagens de erro são desativadas a medida que o usuário progride pelas fases. Na última fase, o usuário terá que mostrar conhecimento do algoritmo de ordenação abordado sem dica ou assistência alguma. Durante o jogo, o desempenho do jogador pode ser avaliado de acordo com a pontuação que ele obtém ao completar as fases. Pedir dicas e cometer erros influenciam negativamente na pontuação, enquanto resolver os desafios apresentados recompensa o jogador com pontos.

Figura 11 – Tela modo Arcade (após desbloquear todas as fases)

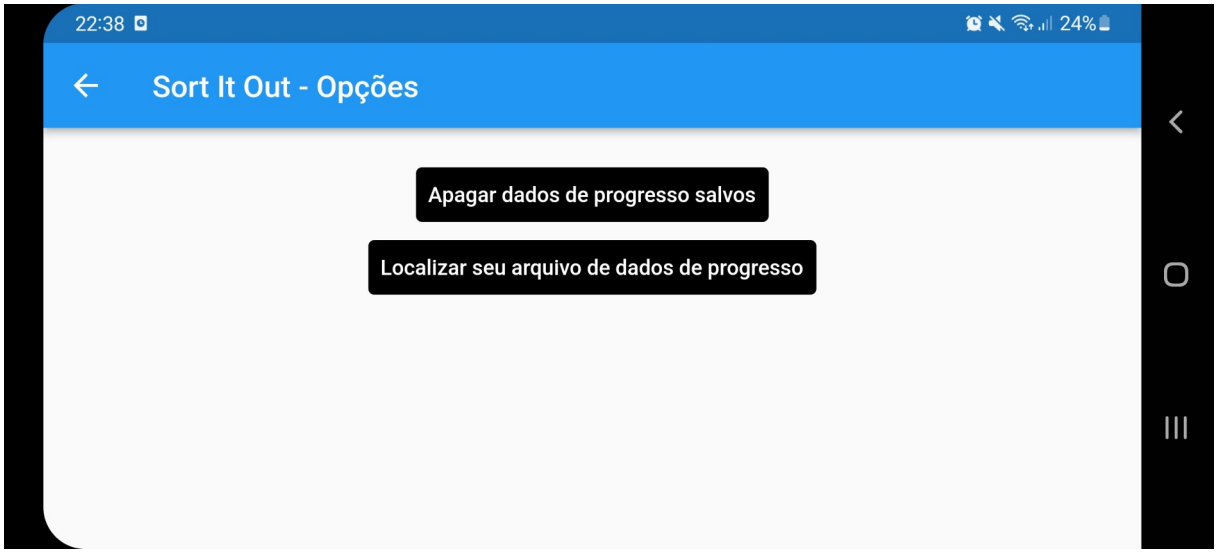


Fonte: Autoria própria

Configurações

No menu de configurações o aluno pode realizar ações úteis como ver a localização do arquivo com os dados salvos do jogador(quais fases foram superadas, quantos pontos o jogador acumulou) além de ter a opção de começar do zero, apagando seu progresso para passar pelas fases novamente.

Figura 12 – Tela de Opções



Fonte: Autoria própria

Regras

Durante a concepção do jogo *Sort it Out*, uma ideia principal era a de manter o jogo o mais intuitivo e simples possível. Jogando alguns dos jogos presentes em trabalhos correlatos, foi percebida a presença de tutoriais bem longos, o que foi evitado completamente no *Sort it Out*.

Dessa forma, as regras são simples:

Algoritmo 1: *Bubble Sort*

O jogador realiza movimentos arrastando os elementos da listagem. A qualquer momento, apenas dois elementos estarão “selecionados” representados pela cor verde. Estes elementos devem ser ordenados, seguindo a ordem do menor elemento à esquerda. Além disso, se os elementos já estiverem na ordem correta, o jogador deve pressionar o botão “Passar” para prosseguir com a fase sem fazer mudanças na ordenação.

Ao errar o movimento, o jogador perde pontos, em quantidade maior de acordo com a dificuldade da fase, quanto mais difícil a fase, mais pontos são perdidos ao errar. Acertando o movimento, o jogador recebe pontos, sendo essa quantidade de pontos ganhos fixa, em contraponto à quantidade de pontos perdidos.

A fase é considerada completa ao ordenar completamente a listagem, tendo todos os elementos na posição correta.

Figura 13 – Fase 1 do algoritmo *Bubble Sort*

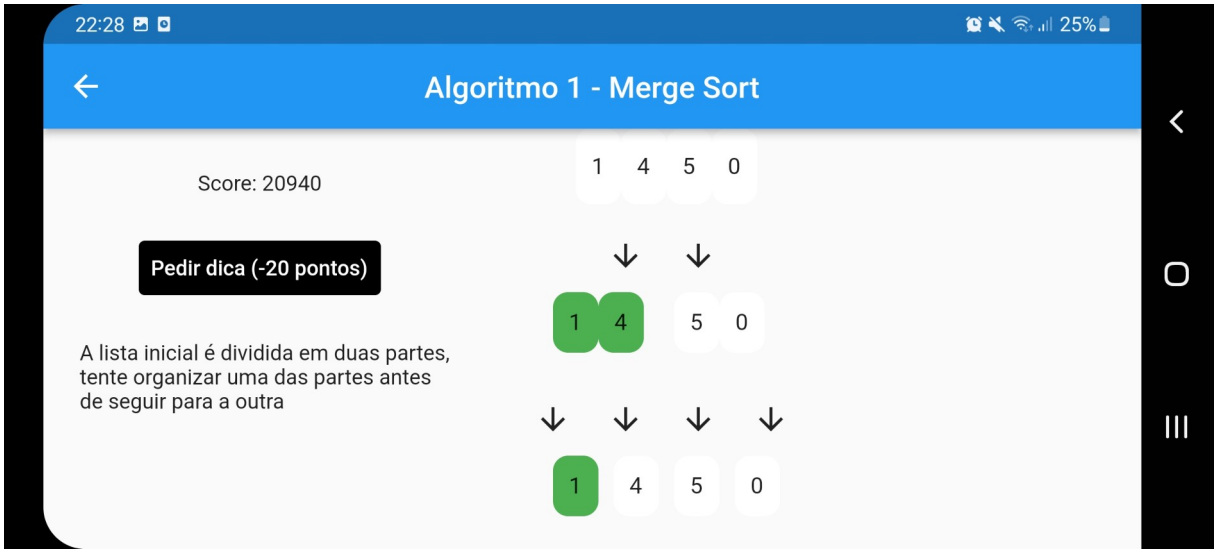
Fonte: Autoria própria

Algoritmo 2: *Merge Sort*

Nesse algoritmo o jogador tem dois tipos de fases:

– Nas fases de numeração ímpar (fases 1 e 3): O jogador tem à sua disposição o algoritmo inteiro estruturado de maneira didática em estilo de “árvore”, sendo seu objetivo definir em que ordem os passos da ordenação acontecem, de acordo com o funcionamento do algoritmo recursivo *Merge Sort*.

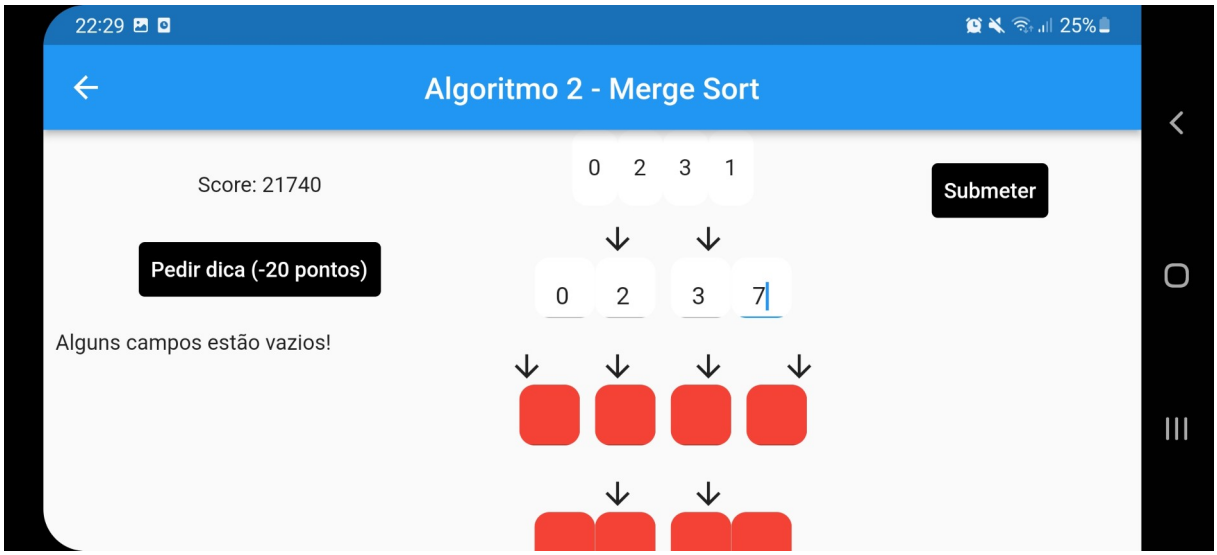
Figura 14 – Fase 1 do algoritmo *Merge Sort*



Fonte: Autoria própria

– Nas fases de numeração par (fases 2 e 4): O jogador tem a estrutura básica do algoritmo pronta, porém com os elementos não preenchidos, sendo o objetivo do jogador preencher os elementos um por um de forma correta de acordo com o algoritmo *Merge Sort*, fazendo a submissão de sua tentativa após preencher todos os campos, sabendo se errou ou não após a submissão.

Figura 15 – Fase 2 do algoritmo *Merge Sort*



Fonte: Autoria própria

5.2 DESENVOLVIMENTO DO SOFTWARE

Tomando como base os documentos criados previamente, foi iniciado o desenvolvimento do aplicativo. Após o *setup* inicial, foi criada uma estrutura para o código, seguindo o padrão de design Provider.

5.2.1 ESTRUTURA UTILIZADA

Utilizando a linguagem *Dart* e a *framework Flutter*, foi criada uma estrutura para o desenvolvimento do aplicativo. A ideia era tornar o desenvolvimento modular, criando uma tela para cada fase de cada modo. Por exemplo, para os estágios do algoritmo *Bubble Sort*, cada fase tem uma classe, que no caso do *Flutter* é um *Widget*, elemento com aparência visual. E para fase também foi criado um *Provider*. O *Provider* é a classe que faz o gerenciamento de estados, mantendo essa lógica fora da classe de visualização.

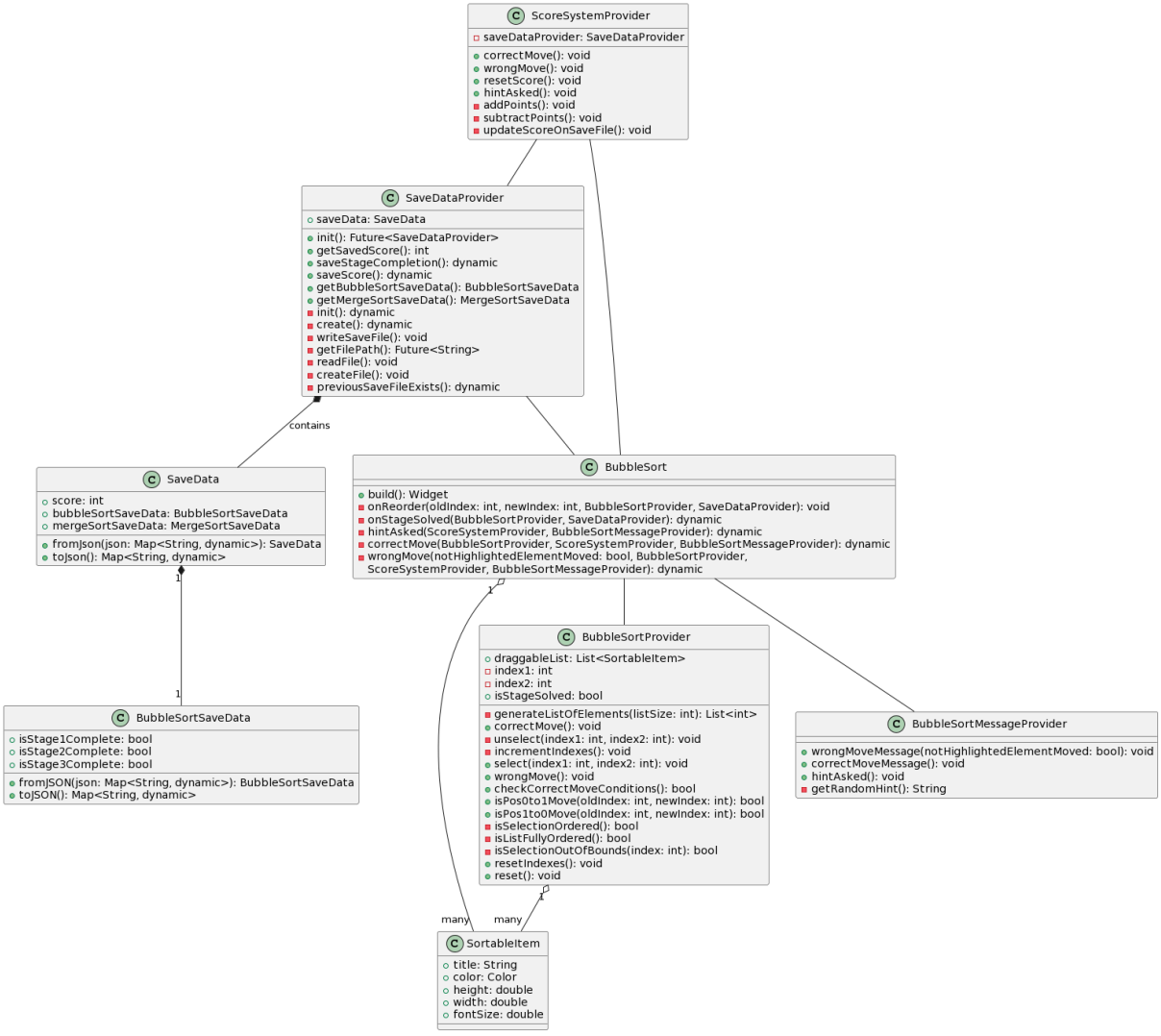
No *Flutter* os estados não são mutáveis, o que faz com que cada vez que alguma instância de “objeto” precisa sofrer alguma mudança nos seus atributos, ela deve ser substituída por outra instância com os atributos mudados. Por exemplo, cada vez que um elemento precisa mudar da cor verde para a cor vermelha, o elemento de cor verde deve deixar de existir, um elemento de cor vermelha deve ser criado e colocado no lugar do elemento anterior.

Tanto para manter a informação de quantos pontos o jogador arrecadou quanto para registrar que fases o jogador completou, foram criados *Providers* com esses

fins. Essas informações ficam guardadas na memória interna do dispositivo móvel em um arquivo.

Para ilustrar o relacionamento entre as classes principais pode-se observar a figura abaixo:

Figura 16 – Diagrama de classes parcial do aplicativo Sort It Out (somente uma fase)



Fonte: Autoria própria

Os pontos mais importantes são que os *Providers* são únicos, portanto não são instanciados várias vezes durante a execução do aplicativo, além disso eles são todos relacionados fortemente com as telas, nomeadas simplesmente pelo nome do algoritmo que representam. Todas as lógicas de gerar elementos, adicionar pontos, mudar a mensagem exibida, são executadas nos *Providers*. As telas apenas exibem os *Widgets*, elementos com a parte visual do aplicativo, invocando o *Provider* para

as demais funções. Os *Models* são contidos pelos *Providers* e pelas telas, já que os *Providers* os geram e as telas os exibem.

Vale salientar que no diagrama algumas informações foram omitidas para fim de tornar viável sua visualização e esse diagrama, além de um diagrama mais detalhado, estão presente no repositório do aplicativo, que pode ser acessado pelo link: https://github.com/igorglatz/sort-it-out.

O diagrama foi feito com o PlantUML, uma ferramenta *open source* baseada em texto de desenho de diagramas que usa uma linguagem própria para a definição dos diagramas (ROQUES, 2006).

5.3 PROBLEMAS E SOLUÇÕES ENCONTRADAS

Inicialmente, o problema encontrado foi tomar uma decisão sobre que arquitetura utilizar para o desenvolvimento do aplicativo. Após pesquisa online, não parecia haver um consenso sobre que padrões de *Design* são usados no desenvolvimento de aplicativos *Flutter*, tanto aplicativos comerciais comuns quanto jogos. A solução foi a criação da estrutura anteriormente apresentada, com *Providers*, *Models* e Telas.

Outra dificuldade encontrada bem no início do desenvolvimento foi adaptar os algoritmos apresentados (*Bubble Sort* e *Merge Sort*) para que eles fossem desconstruídos passo a passo e os alunos pudessem interagir em tempo real com esses algoritmos. Como esses algoritmos são realizados muito rapidamente em situações normais, era necessária uma lógica para que o aplicativo esperasse uma entrada do aluno a cada passo do algoritmo. Para esse fim, foram criadas as interfaces interativas, sendo que no algoritmo *Bubble Sort* essa interface é uma listagem com *drag and drop* e no caso do *Merge Sort*, foram criadas interfaces utilizando botões para fazer a interação com o aluno em um tipo de fase e caixas de texto no outro tipo.

Um problema apresentado pela *framework* junto à linguagem *Dart*, foi que as classes *StatefulWidget*, elementos visuais que possuem estado, não podem ter seus atributos modificados após serem instanciados, o que é permitido em outras linguagens tradicionais como Java, por exemplo. Essa limitação dificultou o processo de fazer os elementos trocarem de cor, e exigiu que cada vez que esses elementos

precisassem mudar algum de seus atributos, todo o elemento devesse ser instanciado novamente, destruindo o antigo. Além disso as listas desses elementos não notificam as classes que as usam, a não ser que toda a listagem seja instanciada novamente também, criando uma situação onde o desenvolvedor tem que criar uma lista, adicionar todos os elementos que já existiam nessa lista nova, e substituir a lista antiga pela nova recém-criada.

6 AVALIAÇÃO

Visto que o intuito do aplicativo é auxiliar no ensino de algoritmos de ordenação na disciplina de Estruturas de Dados, foi realizado um teste, para que fossem obtidas métricas sobre o uso do aplicativo em sala de aula.

Dessa forma, foi selecionado o método MEEGA+, previamente mencionado, como ferramenta de medição de aspectos do uso do aplicativo. Mais especificamente, foi utilizado o formulário para alunos do método, que possui perguntas a serem respondidas pelos alunos após a utilização do jogo.

Como os alunos a serem questionados são alunos da Universidade Federal de Santa Catarina, foi tomado como base o questionário em português disponibilizado no site do GQS – Grupo de Qualidade de Software, do qual os autores do MEEGA+ fazem parte.

6.1 MEEGA+

Conforme previamente mencionado, o MEEGA+ é um modelo para avaliar a qualidade de jogos educacionais em termos de usabilidade e experiência do jogador, da perspectiva do jogador no contexto de educação em computação (PETRI et al., 2019).

O modelo se baseia na medição de certas dimensões para avaliar a qualidade do software criado, sendo essas dimensões divididas em duas categorias: Usabilidade e Experiência de Jogador. Na categoria de Usabilidade, se encontram as dimensões de Estética, Capacidade de aprendizado, Operabilidade e Acessibilidade. Já na categoria de Experiência de Jogador, são englobadas as

dimensões de Proteção de Erro do Usuário, Atenção Focada, Diversão, Desafio, Interação Social, Confiança, Relevância, Satisfação e Aprendizagem Percebida.

Dessa forma, são aplicados questionários, que possuem questões relacionadas a cada uma dessas dimensões, que são aplicados com os alunos. O questionário permite a adição também de perguntas relacionadas a objetivos específicos do jogo analisado. No caso do aplicativo Sort it out, o objetivo é o auxiliar no ensino de algoritmos de ordenação. O formulário dá a oportunidade de inserir mais de uma questão nessa seção, porém foi decidido somente adicionar uma pergunta relacionada ao objetivo principal, para não aumentar significativamente o número de perguntas.

6.2 EXECUÇÃO

A execução da avaliação foi realizada em sala de aula, na disciplina de Estruturas de Dados, no curso de Sistemas de Informação, na Universidade Federal de Santa Catarina. Os participantes foram 15 alunos da disciplina, todos do sexo masculino e com idade entre 18 e 29 anos. Foi feita uma apresentação simples com poucos slides explicando superficialmente o objetivo de cada fase, além da motivação inicial para a criação do aplicativo e características do seu desenvolvimento. Também foi dada uma breve introdução sobre o que é o MEEGA+.

A aplicação do jogo aconteceu perto do final do primeiro semestre letivo de 2023, o que significa que os alunos já tinham conhecido os algoritmos presentes no jogo, agilizando a explicação do aplicativo. A premissa inicial era explicar os algoritmos pela primeira vez logo antes de aplicar o jogo, porém foi aproveitada essa oportunidade para avaliar como o jogo se comporta em nível de revisão de conceitos aprendidos a um tempo atrás.

O tempo de aula previsto era de 90 minutos, com a apresentação inicial sendo de mais ou menos dez minutos e um tempo de em torno de 15 a 20 minutos para preparar o uso do aplicativo nos celulares dos alunos. O aplicativo foi distribuído pelo *Moodle* da disciplina, onde os alunos baixaram o pacote *Android* (APK) da aplicação e o instalaram em seus dispositivos. Após o uso do aplicativo por em torno de 30 minutos, os alunos responderam ao Questionário do Aluno em formato de um

formulário *Google* (*Google Forms*), com todas as questões do questionário original reproduzidas digitalmente, também disponibilizado aos alunos pelo *Moodle* da disciplina.

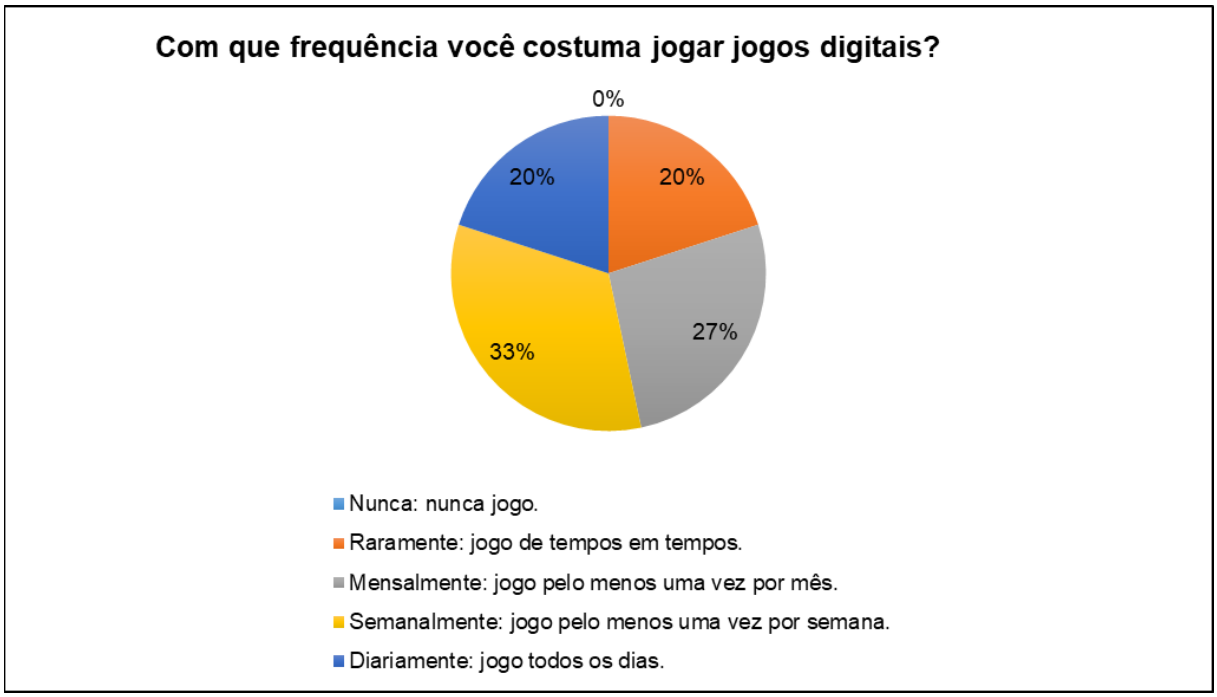
Para gerar tabelas e gráficos dos resultados da avaliação, foi utilizada a planilha disponibilizada juntamente ao Questionário do Aluno do MEEGA+, inserindo os dados na planilha de acordo com as instruções também disponibilizadas no site do GQS (http://www.gqs.ufsc.br/quality-evaluation/meega-plus/) grupo ao qual os autores do MEEGA+ pertencem (PETRI et al., 2019).

6.3 RESULTADOS

Após o preenchimento dos formulários pelos alunos, foram feitas as alterações necessárias de acordo com a planilha auxiliar disponibilizada para o MEEGA+, onde foram atribuídos às perguntas com conceitos como “Discordo totalmente” a “Concordo totalmente” números de -2 a 2, dependendo do quão negativa ou positiva é a resposta. A seguir são apresentados alguns gráficos gerados de acordo com as respostas recolhidas.

Começando com a seção de informações demográficas, conforme mencionado anteriormente, todos os alunos são do sexo masculino, tendo entre 18 e 29 anos de idade. Considerando essa faixa etária e o fato de que são todos alunos de Sistemas de Informação, pode-se observar que nenhum dos alunos são completamente inexperientes com jogos digitais. Além disso, 53% dos alunos jogam semanalmente ou diariamente, o que significa um alto nível de familiaridade desses alunos com jogos digitais.

Figura 17: Terceira pergunta da seção de Informações Demográficas

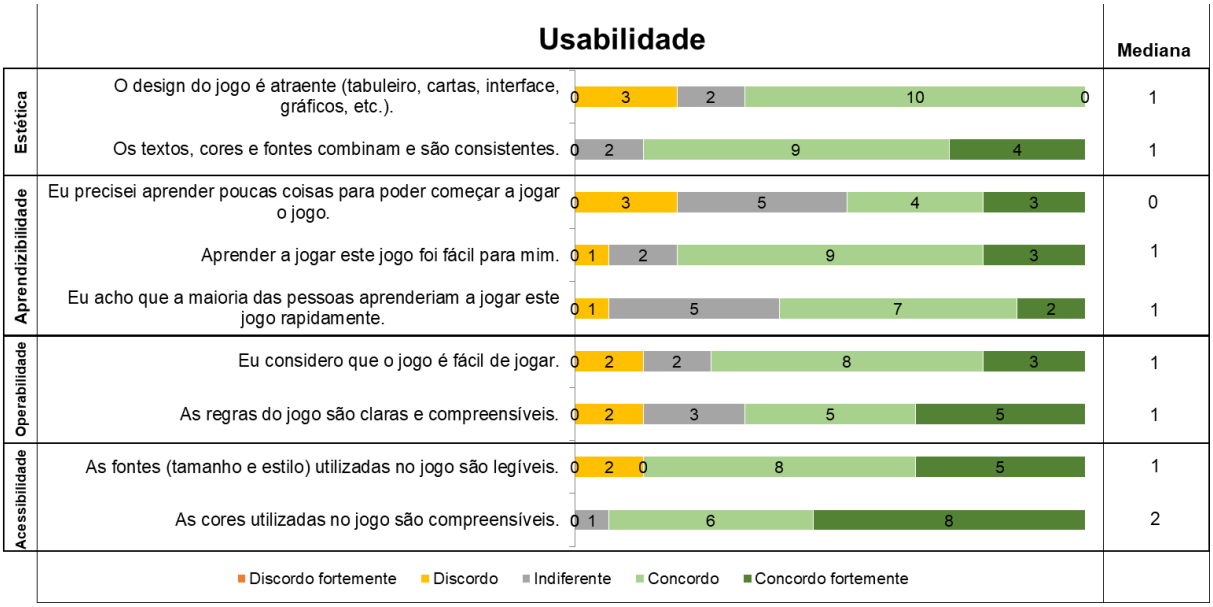


Fonte: Gerada pela planilha auxiliar do MEEGA+, utilizando os dados recolhidos

Na Figura 18, referente às perguntas com o tema Usabilidade, pode-se ver que de forma geral as opiniões foram positivas, com destaque negativo à proposição “Eu precisei aprender poucas coisas para poder começar a jogar o jogo” e demais proposições da subseção de Aprendizibilidade. Um comentário recorrente feito é que não foi explicado com detalhes suficientes o funcionamento do aplicativo e isso causou certa confusão nos alunos, que acabavam resolvendo a mesma fase múltiplas vezes por não saber que deveriam voltar para o menu com as fases após solucionar uma fase, a falta desse redirecionamento para o menu anterior foi um ponto fraco observado por alguns alunos. Além disso, nas fases onde é necessário preencher caixas de texto com números, o teclado numérico muitas vezes ocupava um espaço considerável da tela, ocultando grande parte da interface da fase e dificultando sua solução pelos alunos.

Talvez uma razão para esses problemas de clareza, seja o esforço para não dar informações demais para o aluno inicialmente, para não tornar a resolução fácil demais. Esse esforço pode ter sido exagerado em retrospectiva.

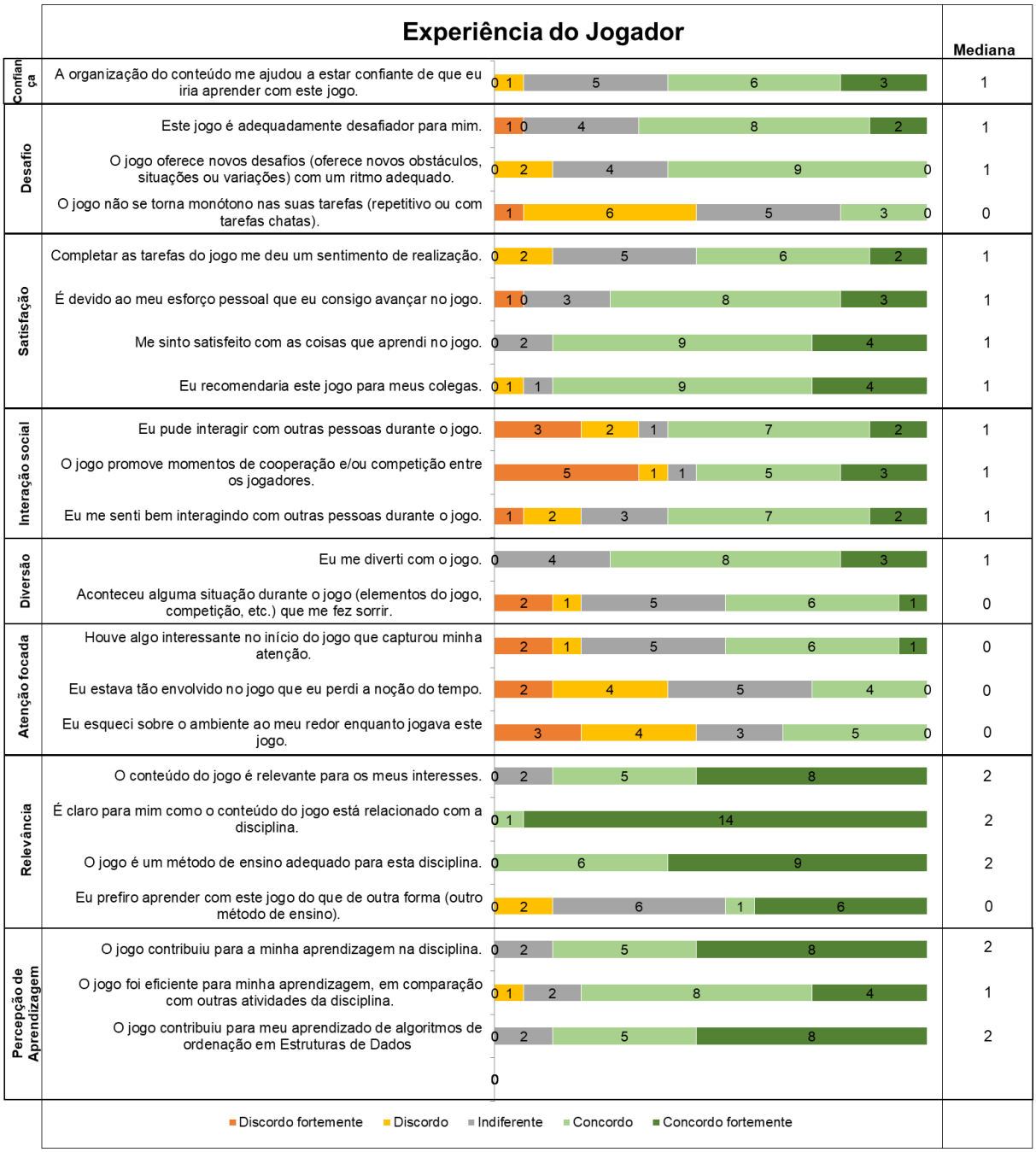
Figura 18: Gráfico de Usabilidade



Fonte: Gerada pela planilha auxiliar do MEEGA+, utilizando os dados recolhidos

Quanto à experiência do jogador, os principais pontos fracos foram de falta de imersão, monotonia nas tarefas (o que pode ser relacionado em parte à natureza repetitiva de executar algoritmos de ordenação) e falta de interação social. Os pontos fortes que se destacam são a relevância e a percepção de aprendizagem.

Figura 19: Gráfico de experiência do jogador



Fonte: Gerada pela planilha auxiliar do MEEGA+, utilizando os dados recolhidos

7 CONSIDERAÇÕES FINAIS

O objetivo principal desse trabalho foi criar um jogo que pode ser utilizado como ferramenta para auxílio na aprendizagem de algoritmos de ordenação na disciplina de Estruturas de Dados. Para esse fim, vários objetivos específicos foram definidos.

Inicialmente, foram sintetizados conceitos como Pensamento Computacional e Jogos Educacionais, a fim de ter um maior entendimento sobre os tipos de jogos educacionais que existem, as técnicas envolvidas em criá-los e conceitos que podem ser aplicados no desenvolvimento de uma ferramenta de aprendizagem nesse caráter.

Foi também introduzida a disciplina de Estruturas de Dados, explicando sua importância no currículo de cursos relacionados a temas de Computação, os algoritmos utilizados e seu papel transicional entre as matérias mais simples dos cursos e matérias mais avançadas.

Foram encontrados diversos exemplos de jogos educativos, tanto em contexto de ensino de Computação, quanto em ensino de outros temas. Alguns exemplos tratavam da disciplina de Estruturas de Dados e serviram como inspiração para a criação de um aplicativo que permitisse a interação de alunos com as Estruturas de Dados que são demonstradas na disciplina. Outros, tratando de outros assuntos de Computação, trouxeram *insights* sobre técnicas de desenvolvimento e avaliação de jogos educativos.

A escolha pela *framework Flutter*, apesar de trazer dificuldades, também permitiu que fosse criado um aplicativo com o potencial de ser multiplataforma, além de possuir fácil compilação e preparação inicial. Com o fato de ser uma framework para aplicativos tanto *Android* (alvo do presente trabalho) quanto *iOS* e *Web*, permite que no futuro, com algumas adaptações e testes para essas plataformas, o aplicativo Sort it Out possa se desenvolver ainda mais.

O desenvolvimento do aplicativo foi repleto de desafios e acabou sendo mais complexo do que tinha sido pensado inicialmente. A barreira de criar algoritmos interativos foi algo difícil de superar, a tecnologia *Flutter* tem algumas limitações em relação a aplicações dessa forma. Talvez um estudo mais amplo de tecnologias diferentes pudesse revelar uma ferramenta mais adequada para o desenvolvimento de um aplicativo dessa categoria.

Durante a pesquisa de trabalhos correlatos a ferramenta MEEGA+ se destacou como uma das mais completas em avaliação da qualidade de jogos educacionais, o que incentivou o seu uso para a avaliação do aplicativo *Sort it Out*.

Ao pôr em prática a avaliação, utilizando o MEEGA+, foram levantados pontos fortes e fracos da aplicação. Entre os destaques positivos se encontram a relevância do aplicativo em relação ao conteúdo aprendido e a percepção de aprendizagem que os alunos tiveram. Já como pontos fracos, a experiência de usuário poderia ter sido mais desenvolvida, com explicações mais detalhadas do funcionamento do jogo, além de alterações menores na interface para melhorar a usabilidade do jogo de forma geral.

Por fim, como melhorias futuras se sugere a adição de mais algoritmos diversificados, com fases possuindo desafios também diversificados, a fim de aumentar o repertório do aplicativo e oferecer uma experiência mais completa dos algoritmos de ordenação em Estruturas de Dados.

Foi um exercício interessante pensar em formas de apresentar algoritmos que são tão comumente apresentados em slides ou quadros em sala de aula manualmente, certamente uma experiência de aprendizado.

REFERÊNCIAS

DICHEVA, Darina; IRWIN, Keith; DICHEV, Christo. **OneUp: Engaging Students in a Gamified Data Structures Course**. In: ACM TECHNICAL SYMPOSIUM ON COMPUTER SCIENCE EDUCATION, 50., 2019, Nova Iorque. 50th ACM Technical Symposium on Computer Science Education (SIGCSE '19). Nova Iorque: Association For Computing Machinery, 2019. p. 383-392.

DICHEVA, Darina; HODGE, Austin. **Active Learning through Game Play in a Data Structures Course**. In: ACM TECHNICAL SYMPOSIUM ON COMPUTER SCIENCE EDUCATION, 49., 2018, Nova Iorque. Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18). Nova Iorque: Association For Computing Machinery, 2018. p. 834-839.

OKADA, Alexandra; SHEEHY, Kieron. **O VALOR DA DIVERSÃO NA APRENDIZAGEM ON-LINE: um estudo apoiado na pesquisa e inovação responsáveis e dados abertos**. Revista E-Curriculum, [S.L.], v. 18, n. 2, p. 590-613, 26 jun. 2020. Pontificia Universidade Catolica de Sao Paulo (PUC-SP). http://dx.doi.org/10.23925/1809-3876.2020v18i2p590-613. Disponível em: https://revistas.pucsp.br/curriculum/article/view/48014. Acesso em: 26 abr. 2021.

BATTISTELLA, Paulo; VON WANGENHEIM, Christiane Gresse. **Games for Teaching Computing in Higher Education: a systematic review**. IEEE Technology and Engineering Education (ITEE) Journal. Florianópolis, p. 8-30. 29 mar. 2016.

DAGHESTANI, Reem S. Al-Towirgi Lamya F.; IBRAHIM, Lamiaa F.. **Increasing Students Engagement in Data Structure Course Using Gamification. International Journal Of E-Education, E-Business, E-Management And E-Learning**, Taif, v. 8, n. 4, p. 193-211, 08 mar. 2018. International Academy Publishing (IAP). http://dx.doi.org/10.17706/ijeeee.2018.8.4.193-211.

MARTINS, Amilton Rodrigo de Quadros; ELOY, Adelmo Antonio da Silva. **EDUCAÇÃO INTEGRAL POR MEIO DO PENSAMENTO COMPUTACIONAL: letramento em programação: relatos de experiência e artigos científicos**. Curitiba: Appris, 2019. 363 p.

WING, J. **PENSAMENTO COMPUTACIONAL Um conjunto de atitudes e habilidades que todos, não só cientistas da computação, ficaram ansiosos para aprender e usar**. Revista Brasileira de Ensino de Ciência e Tecnologia , v. 9, n. 2, 2016. Disponível em: http://dx.doi.org/10.3895/rbect.v9n2.4711.

BECKER, Katrin. **What's the difference between gamification, serious games, educational games, and game-based learning?** Academia Letters, [S.L.], v. 1, n. 1, p. 1-4, 27 jan. 2021. Academia.edu. http://dx.doi.org/10.20935/al209.

ROJAS-SALAZAR, Alberto; RAMÍREZ-ALFARO, Paula; HAAHR, Mads. **Learning Binary Search Trees Through Serious Games**. First International Computer

Programming Education Conference (Icpec 2020), Daghstul, Alemanha, v. 81, n. 22, p. 1-7, jun. 2020. Disponível em: https://drops.dagstuhl.de/opus/volltexte/2020/12309. Acesso em: 29 set. 2021.

OLIVEIRA, Placida et al. **Jogo de RPG para o Desenvolvimento de Habilidades do Pensamento Computacional no Ensino Fundamental.** In: WORKSHOP SOBRE EDUCAÇÃO EM COMPUTAÇÃO, 29, 2021, Florianópolis. 2021: Anais do XXIX Workshop sobre Educação em Computação. Florianópolis: Csbc, 2021. v. 29, p. 41-50.

CRAWFOD, Chris. **The Art of Digital Game Design.** Washington State University, Vancouver, 1982

JUUL, Jesper. **Half-Real: Video Games between Real Rules and Fictional Worlds.** The MIT Press, 2005, ISBN: 0262101106

HUIZINGA, J. **Homo ludens: o jogo como elemento da cultura**. 5o. ed. [S.l.]:Perspectiva, 2003. p. 256

LUCCHESE, Fabiano; RIBEIRO, Bruno. **Conceituação de Jogos Digitais**. UNIVERSIDADE ESTADUAL DE CAMPINAS, 2009. Disponível em: https://www.dca.fee.unicamp.br/~martino/disciplinas/ia369/trabalhos/t1g3.pdf Acesso em: 20 dez. 2021.

DA ROSA, Alexandre; DE JESUS, Andreia; IGARASHI, Gabriel Vaz ; PEREIRA, Vinicius Struginski . **Iron Ears: primeiras impressões de um jogo educativo para ensino de estrutura de dados lineares.** *In*: WORKSHOP SOBRE EDUCAÇÃO EM COMPUTAÇÃO (WEI), 28 , 2020, Cuiabá. **Anais** [...]. Porto Alegre: Sociedade Brasileira de Computação, 2020 . p. 31-35. ISSN 2595-6175. DOI: https://doi.org/10.5753/wei.2020.11124.

ROSA, Yuri Kaiser da; COELHO, Everton Schafaschek. **Space Code: Um Jogo Para O Desenvolvimento De Pensamento Computacional Utilizando Interfaces Tangíveis**. 2018. 151 f. TCC (Graduação) - Curso de Sistemas de Informação, Centro Tecnológico, Universidade Federal de Santa Catarina, Florianópolis, 2018. Cap. 1.

BATTISTELLA, Paulo E. et al. **SORTIA 2.0: A sorting game for data structure teaching**. In: BRAZILIAN SYMPOSIUM ON INFORMATION SYSTEMS, 12., 2016, Porto Alegre. Proceedings of the XII Brazilian Symposium on Information Systems. Porto Alegre: Brazilian Computer Society, 2016. p. 558-565.
BATISTELLA, P. E., Wangenheim, A. von e Wangenheim, C. G. Von. 2012. S**ORTIA - Um Jogo para Ensino de Algoritmo de Ordenação: Estudo de caso na Disciplina de Estrutura de Dados**. 23º Simpósio Brasileiro de Informática na Educação. Rio de Janeiro, Brasil.

SENA, Alexandre da Costa et al. **UMA PROPOSTA DE JOGO EM DUAS ETAPAS PARA CONHECER A COMPUTAÇÃO**. In: MARTINS, Ernane Rosa (org.). Digital Games and Learning. Ponta Grossa: Atena, 2019. Cap. 8. p. 87-102.

DAMASCO, Rafael Euclides. **RETRAINING: UMA SOLUÇÃO COMPUTACIONAL PARA APOIAR O ENSINO DE ESPECIFICAÇÃO DE REQUISITOS**. 2021. 159 p. Trabalho de conclusão de curso (Graduação em Sistemas de Informação) - Universidade Federal de Santa Catarina, Florianópolis, 2021. Disponível em: https://repositorio.ufsc.br/bitstream/handle/123456789/228407/TCC.pdf?sequence=1&isAllowed=y. Acesso em: 31 jan. 2022.

SOMMERVILLE, I. **Engenharia de Software**. 9ª. Ed. São Paulo: PEARSON Addison Wesley, 2011

Choi, J. e Hannafin, M. 1995. **Situated Cognition and Learning Environments: Roles, Structures and Implications for Design**. Educational Technology Research and Development, 43(2), p. 53-69.

VEUGEN, Connie. **Adventure Games: Here Be Dragons: advent and prehistory of the Adventure game**. TMG Games, [s. l.], p. 77-99, 2004

PICCOLO, Gustavo Martins. **O universo lúdico proposto por Caillois**. Educación Física y Deportes, Buenos Aires, v. 127, n. 13, p. 1-9, dez. 2008. Anual. Disponível em: http://www.efdeportes.com/efd127/o-universo-ludico-proposto-por-caillois.htm. Acesso em: 09 fev. 2022.

CHINN, Donald; PRINS, Phil; TENENBERG, Josh. **The Role Of The Data Structures Course In The Computing Curriculum**. Journal of Computing Sciences in Colleges, Evansville, v. 19, n. 2, p. 91-93, 2003.

CC2020 TASK FORCE. **Computing Curricula 2020: Paradigms for Global Computing Education**. Nova Iorque: Association for Computing Machinery, 2020. 205 p. ISBN 978-1-4503-9059-0. DOI https://doi.org/10.1145/3467967. Disponível em: https://dl.acm.org/doi/book/10.1145/3467967. Acesso em: 15 fev. 2022.

DE LUCCA, José Eduardo. **Plano de Ensino INE5609 - Estruturas de dados.** Universidade Federal de Santa Catarina. Centro Tecnológico. Departamento de Informática e Estatística, Florianópolis, 14 jul. 2017.

LIMA, Thamyla Maria de Sousa et al. **Edubot: um estudo prático de aprendizagem baseada em problemas no contexto de agentes inteligentes e jogos sérios**. Revista Tecnologias na Educação: Edição Temática IX– III Simpósio Nacional de Tecnologias Digitais na Educação (III-SNTDE), [s. l.], ano 10, v. 27, 2018. Disponível em: https://tecedu.pro.br/wp-content/uploads/2018/11/Art6.Vol27-Ed.Tem%C3%A1ticaIX-Nov-2018.pdf. Acesso em: 22 fev. 2022.

ARIMOTO, Maurício M.; CRUZ, José Henrique R.. **Ensino de Lógica e Programação no Ensino Médio por meio de uma Abordagem Lúdica e Gamificada**. In: WORKSHOP SOBRE EDUCAÇÃO EM COMPUTAÇÃO (WEI), 28. , 2020, Cuiabá. Anais [...]. Porto Alegre: Sociedade Brasileira de Computação, 2020 . p. 166-170. ISSN 2595-6175. DOI: https://doi.org/10.5753/wei.2020.11151.

Ministério da Educação. 2012. **Diretrizes Curriculares Nacionais para Cursos de Graduação em Computação**. Parecer CNE/CNS 136/2012. Disponível em: <http://portal.mec.gov.br>. Acesso em: 24 fevereiro, 2022.

BLIKSTEIN, Paulo. **O pensamento computacional e a reinvenção do computador na educação**. In: www.blikstein.com. [S. l.], 22 dez. 2008. Disponível em: http://www.blikstein.com/paulo/documents/online/ol_pensamento_computacional.html. Acesso em: 2 mar. 2022.

PETRI, Giani; WANGENHEIM, Christiane Gresse von; BORGATTO, Adriano Ferreti. **MEEGA+: Um Modelo para a Avaliação de Jogos Educacionais para o ensino de Computação**. Revista Brasileira de Informática na Educação, [S.l.], v. 27, n. 03, p. 52-81, dez. 2019. ISSN 2317-6121. Disponível em: <https://br-ie.org/pub/index.php/rbie/article/view/v27n035281>. Acesso em: 02 mar. 2022. doi:http://dx.doi.org/10.5753/rbie.2019.27.03.52.

SAVI, Rafael; VON WANGENHEIM, Christiane Gresse; BORGATTO, Adriano Ferreti. **A Model for the Evaluation of Educational Games for Teaching Software Engineering**. 2011 25Th Brazilian Symposium On Software Engineering, [S.L.], v. 25, n. 27, p. 194-203, set. 2011. IEEE. http://dx.doi.org/10.1109/sbes.2011.27.

WING, Jeannette M. **Computational thinking and thinking about computing**. Philosophical Transactions Of The Royal Society A: Mathematical, Physical and Engineering Sciences, [S.L.], v. 366, n. 1881, p. 3717-3725, 31 jul. 2008. The Royal Society. http://dx.doi.org/10.1098/rsta.2008.0118.

HAAS, John K. Abstract. In: HAAS, John K. **A History of the Unity Game Engine**. Orientador: Brian J. Moriarty. 2014. Trabalho de conclusão de curso (Bacharelado em Ciência da Computação) - Worcester Polytechnic Institute, Worcester, 2014. p. 1. Disponível em: https://digital.wpi.edu/concern/student_works/tx31qh96p?locale=en. Acesso em: 7 mar. 2022.

Tecnológico de Monterrey. **Canvas de Diseño**. [2016?] In: Nuevo León, México.. [S. l.]. Disponível em: https://innovacioneducativa.tec.mx/es/recursos-pedagogicos/estrategias-de-aprendizaje-activo/gamification. Acesso em: 13 set. 2022.

VENÂNCIO, João Vítor Demaria. **Avaliação De Jogo Para Apoio Ao Processo De Ensino-Aprendizagem De Técnica De Especificação De Requisitos De Software.** 2022. 208 f. TCC (Graduação) - Curso de Sistemas de Informação, Centro Tecnológico, Universidade Federal de Santa Catarina, Florianópolis, 2022.

SCHUYTEMA, P. **Design de games uma abordagem prática**. São Paulo: Cengage Learning, 2008. ISBN 978-8522106158.

ROQUES, Arnaud. **Frequently Asked Questions**. In: PlantUML. [S. l.], 2006. Disponível em: https://plantuml.com/faq. Acesso em: 4 jun. 2023.

Apêndices

Apêndice A – *Game Design Document*

GDD - Game Design Document

1 Nome do Projeto

Sort it Out

1.1 Resumo
Plataforma: Android
Jogadores: Um jogador, offline
Gênero: Serious game, Puzzle

2 Highconcept

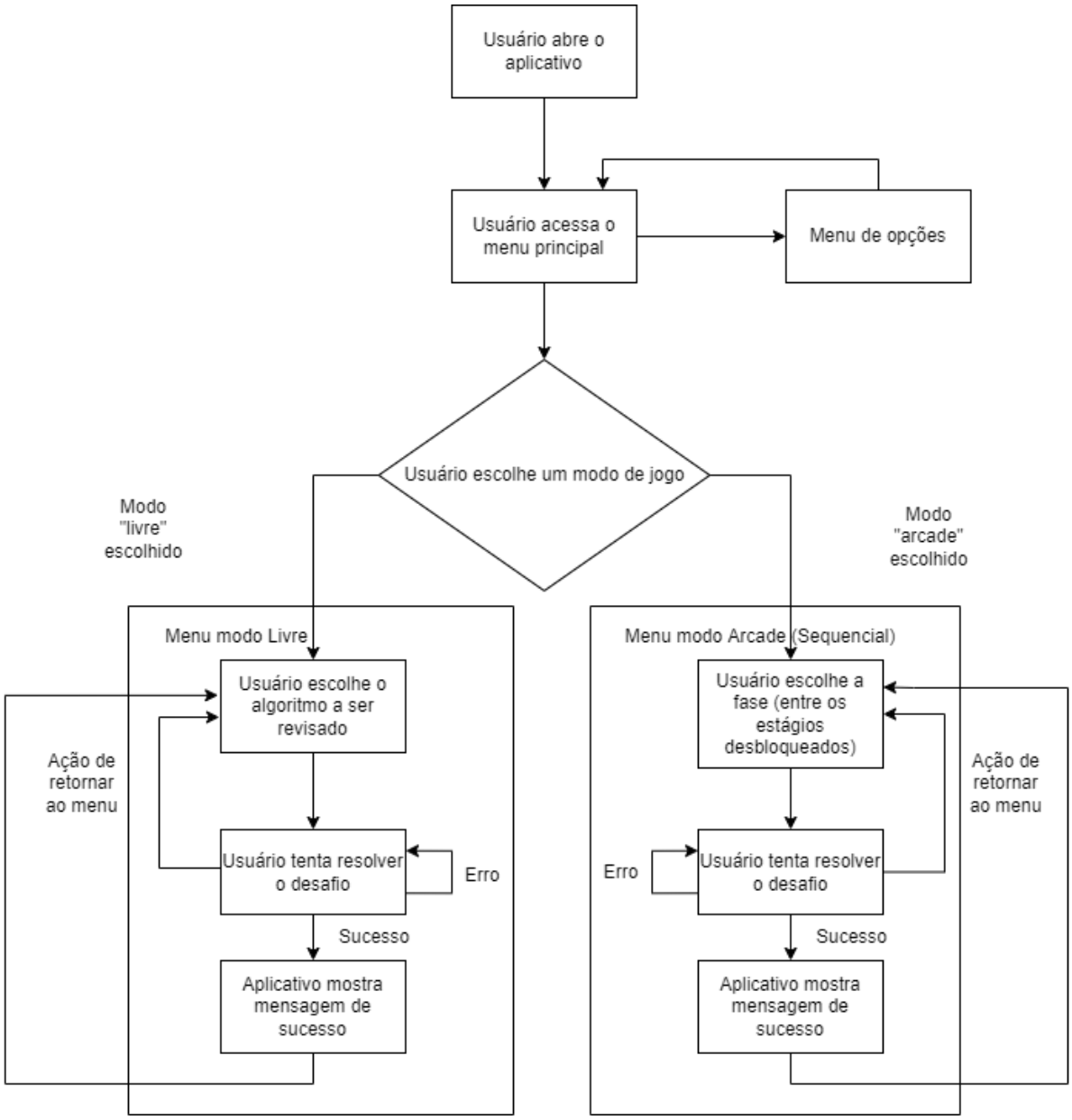
O Sort it Out é um jogo sério educacional do gênero puzzle, que tem o objetivo de auxiliar o processo de aprendizagem na disciplina de Estruturas de Dados, mais especificamente no assunto de Algoritmos de Ordenação. O jogador é desafiado a executar os algoritmos em uma espécie de “teste de mesa” onde os passos do algoritmo são realizados um a um até que a coleção fornecida seja ordenada completamente. Além de ajudar no aprendizado de como os algoritmos funcionam individualmente, também podem ser observados os pontos fortes e fracos de cada algoritmo. Pode ser jogado por adultos de 17 anos para cima, já que é uma ferramenta de aprendizagem com foco no ensino superior de Estruturas de Dados, porém nada impede que haja jogadores mais jovens.

3 Gameplay e Enredo

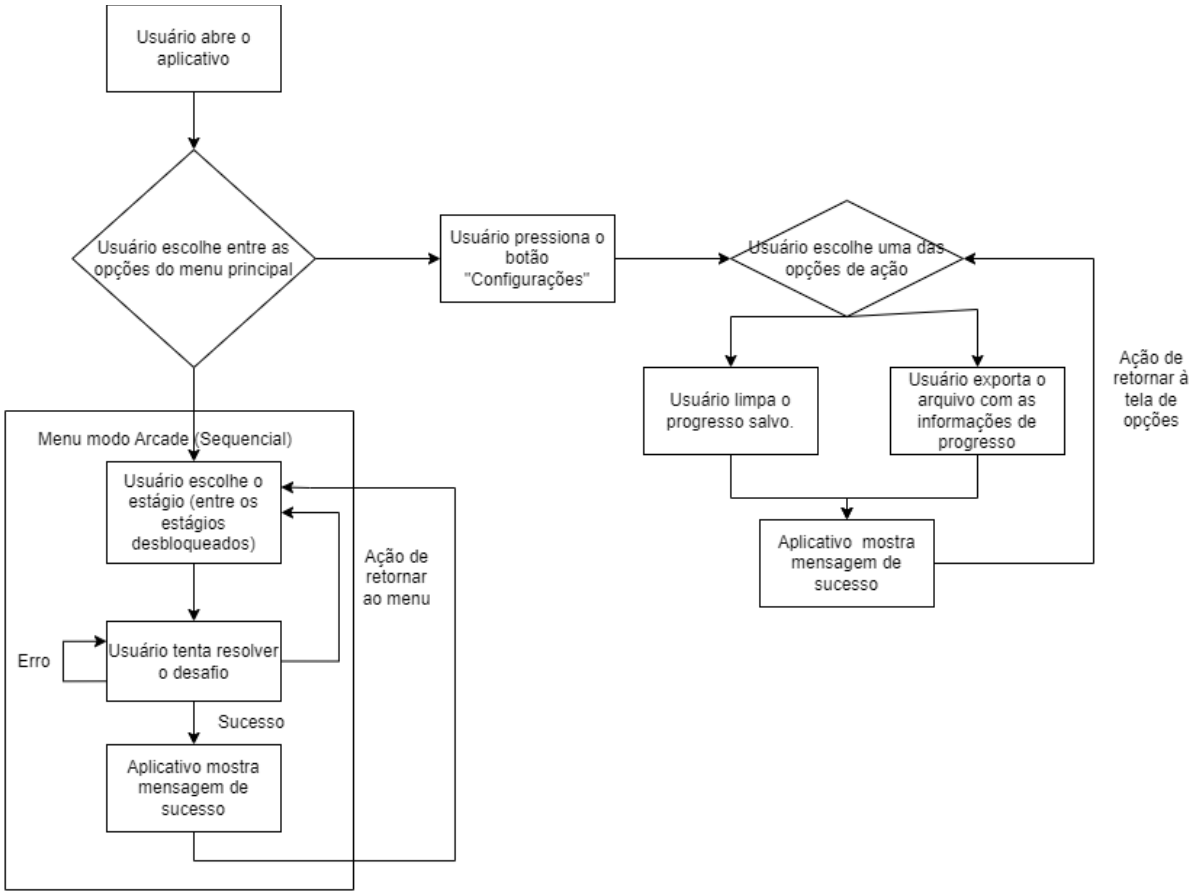
O jogador terá duas opções de modo, uma delas o modo arcade, onde as fases são sequenciais e é necessário respeitar a sequência das fases. A segunda opção é o modo livre, onde o jogador pode escolher um algoritmo já visto, para revisar os conceitos. Escolhendo qualquer um dos dois modos, o jogador deve escolher uma fase que terá um desafio que auxilia no aprendizado de algum conceito. Dessa forma o

jogador continua tentando até obter sucesso, perdendo pontos ao cometer erros ou pedir dicas, tendo como objetivo resolver o desafio com o menor número de erros possível e pedindo o menor número de dicas possível. O jogo não contém tutoriais ou aulas sobre os conceitos apresentados, dessa forma é necessário que esses conceitos sejam explicados previamente por um instrutor/professor.

4 Fluxo do jogo



Versão inicial antes do desenvolvimento

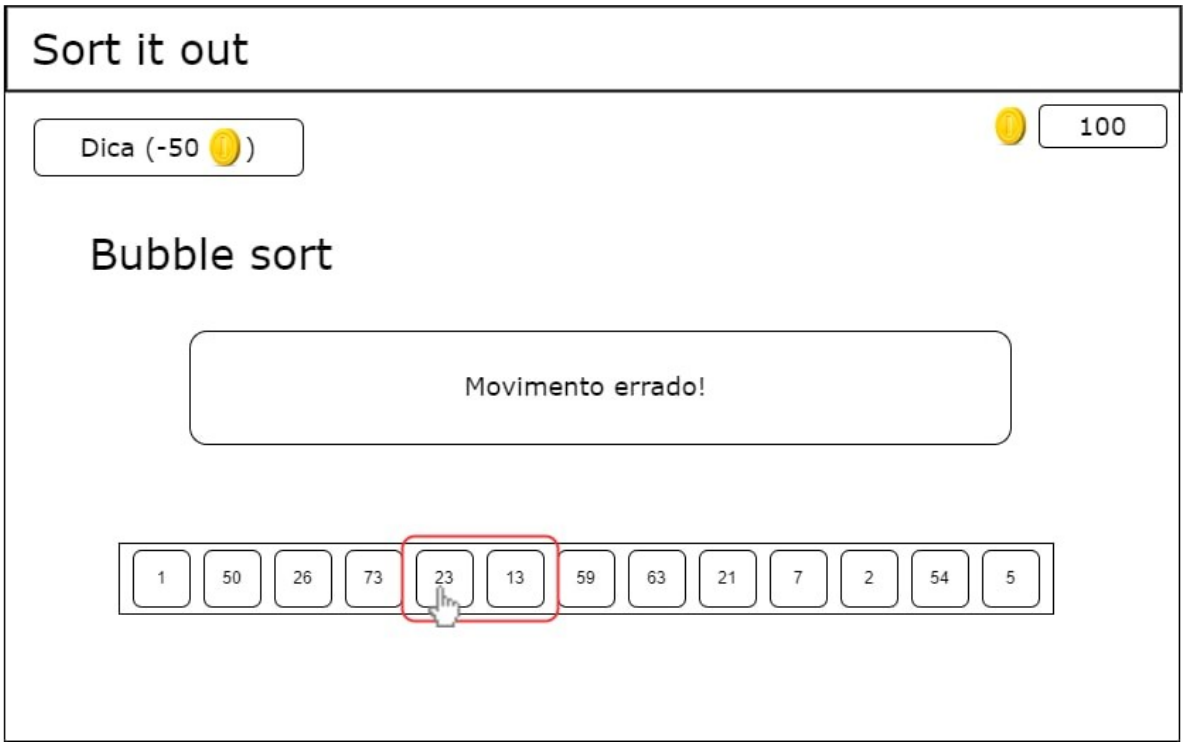


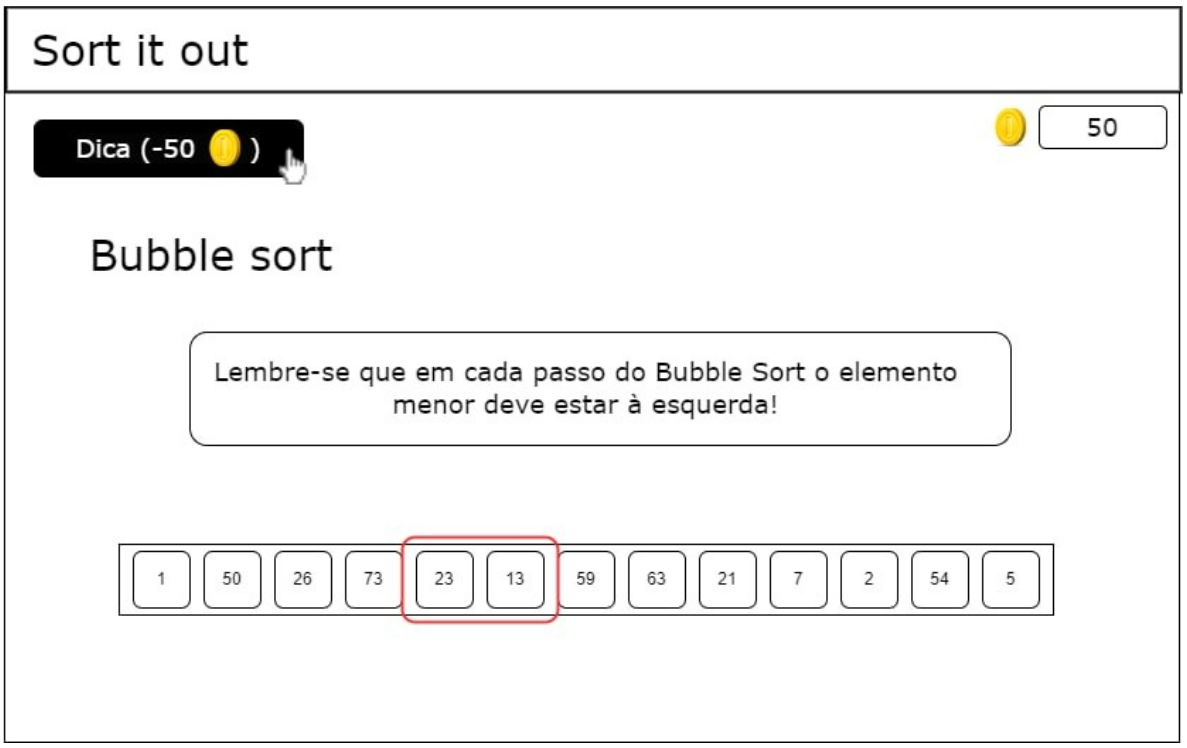
Segunda versão após ajustes feitos durante o desenvolvimento

A qualquer momento o jogador pode voltar ao menu principal. O progresso na fase só é reiniciado no momento que o jogador consegue resolver o desafio proposto. Em cada fase, independente do modo de jogo, o desafio é terminar a ordenação no menor número de passos possível, utilizando o menor número de dicas (quando elas estão disponíveis) e cometendo o menor número de erros possível.

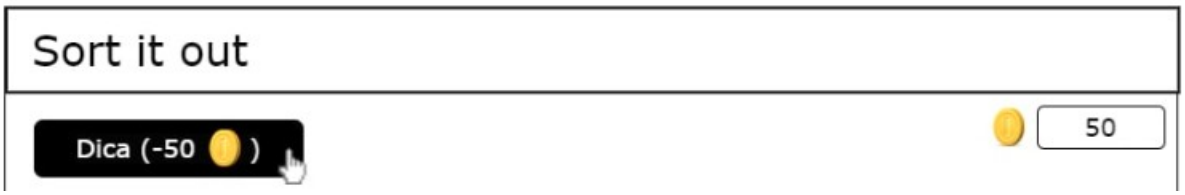
5 Level Design

O design das fases pode ser observado nas imagens abaixo. Há uma representação da estrutura de dados utilizada no problema, sendo que o jogador pode interagir com essa estrutura de dados para fazer a ordenação de seus elementos. Na caixa de texto são mostradas mensagens de erro ou dicas dependendo se o jogador pediu uma dica gastando seus pontos ou não. Em dificuldades maiores não é possível pedir dicas.

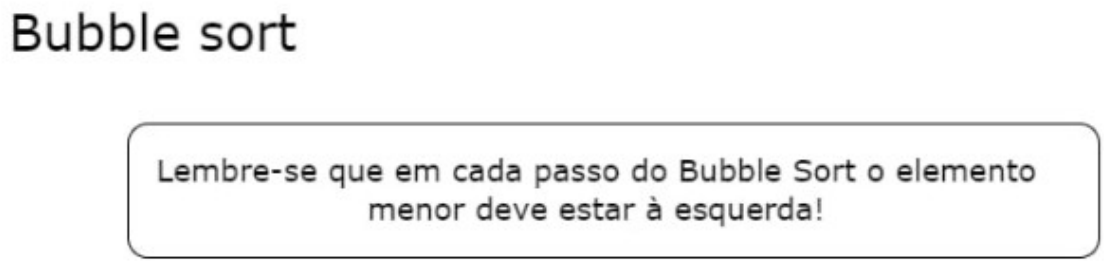

Sort it out
Dica (-50)
100
Bubble sort
Movimento errado!
1
50
26
73
23
13
59
63
21
7
2
54
5


Sort it out
Dica (-50)
50
Bubble sort
Lembre-se que em cada passo do Bubble Sort o elemento menor deve estar à esquerda!
1
50
26
73
23
13
59
63
21
7
2
54
5

6 Interface de usuário



Sistema de Dicas, onde o jogador pode solicitar uma dica sobre a resolução do desafio, gastando uma parte de seus pontos. Não é disponível em todas as dificuldades.



Mensagens de erro que alertam o jogador de erros que foram cometidos e o porquê do movimento ser considerado errôneo.

7 Áudio e música

Descreva as características de áudio e música do jogo.

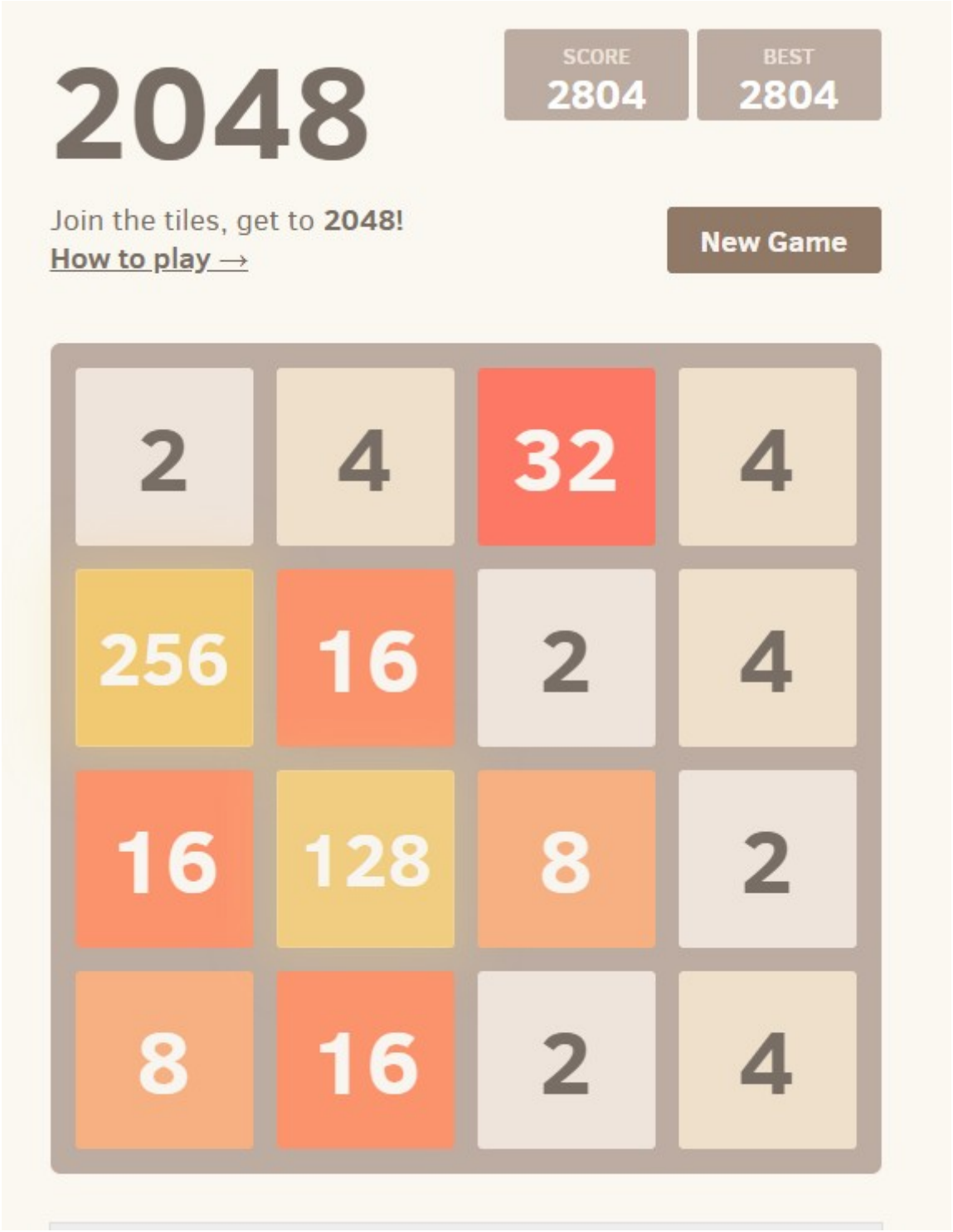
N/A

8 Arte Conceito e referências

Nesta seção, inclua arte conceito ou referências de outros jogos, que ilustram algum aspecto específico ou
amplo deste jogo.

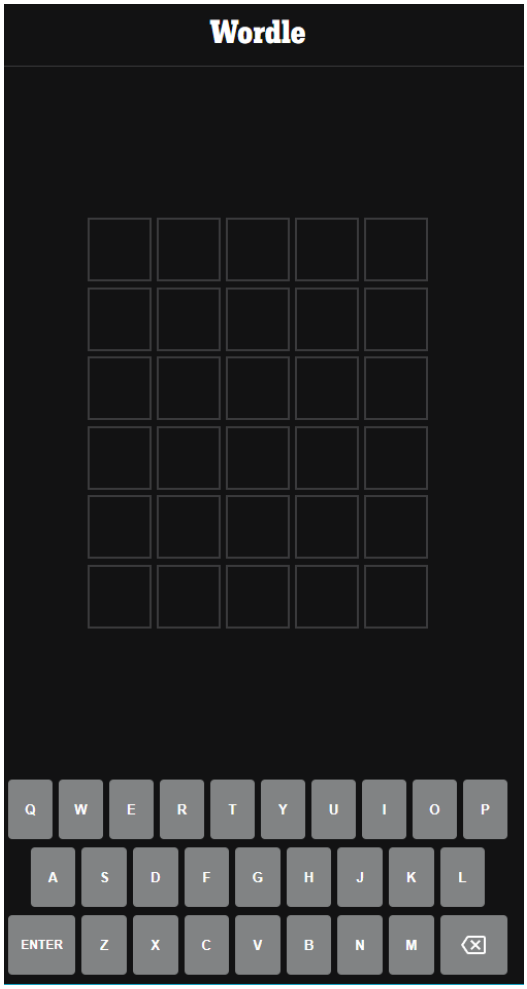
Jogos do gênero puzzle bem simples, com movimentos intuitivos e UI minimalista, inspiraram a tentar fazer um jogo do mesmo gênero e com a mesma simplicidade.

2048



Fonte: https://play2048.co/ Criadora: Gabriele Cirulli, http://gabrielecirulli.com/

Wordle

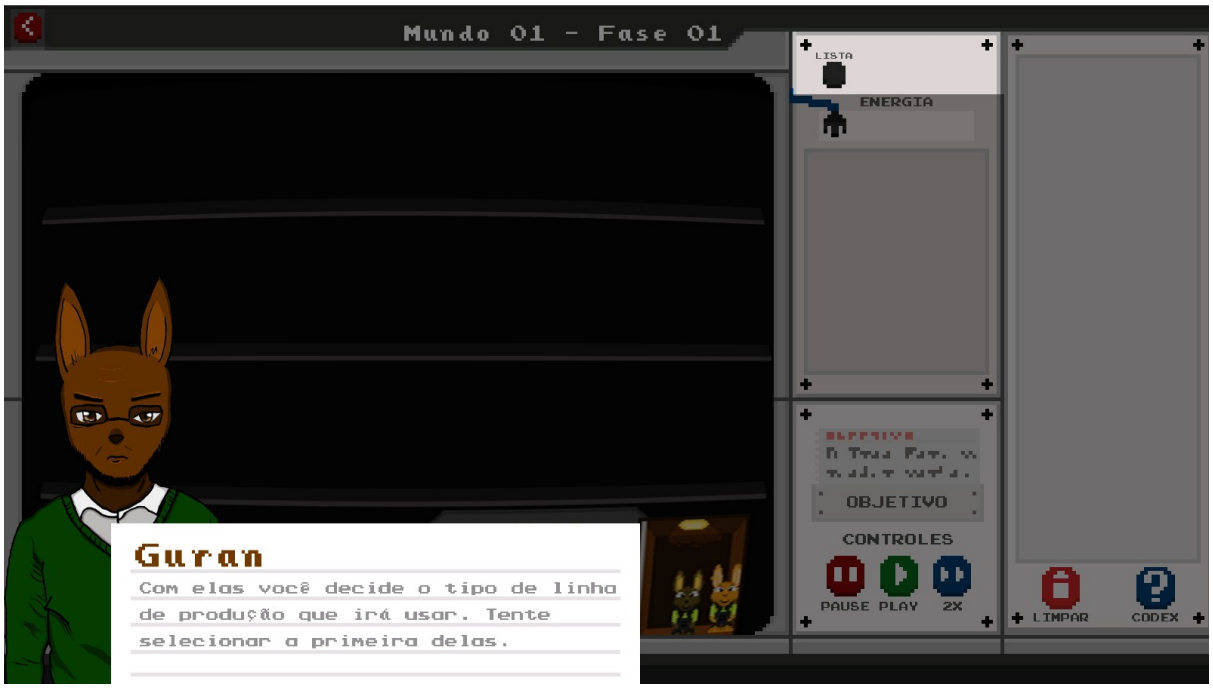


Fonte: https://www.nytimes.com/games/wordle/index.html

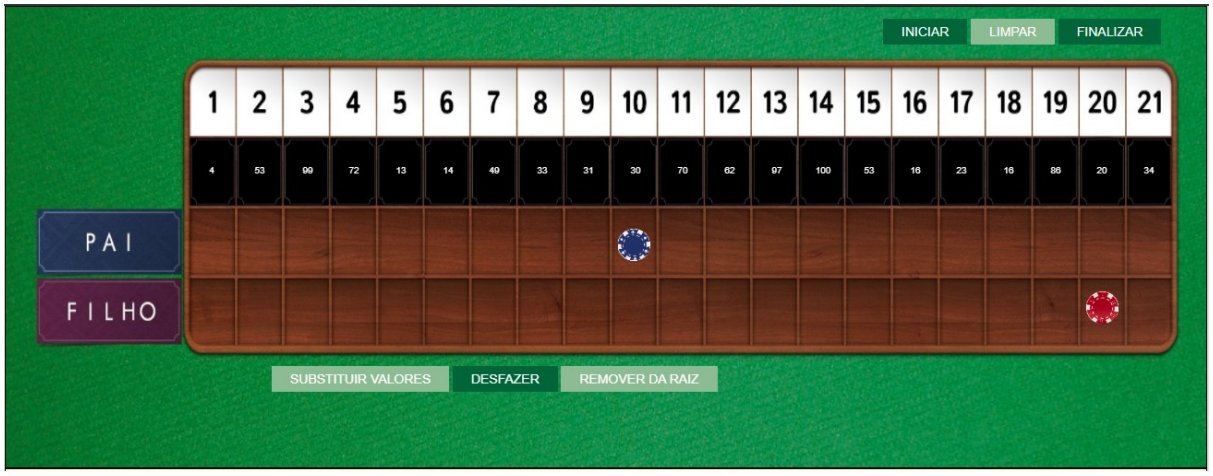
Jogos para aprendizagem em Estruturas de Dados

Jogos que auxiliam no aprendizado de Estruturas de dados, inspiraram a tentativa de criar um jogo que se baseasse na mesma disciplina curricular, porém com controles mais intuitivos do que inserir comandos ou preencher campos de texto.

Iron Ears



Sortia 2.0



Apêndice B – Código fonte

bubble_sort_1.dart

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'package:sort_it_out/src/game_modes/arcade/stages/bubble_sort/
bubble_sort_message_provider.dart';
import 'package:sort_it_out/src/game_modes/arcade/stages/bubble_sort/stage_1/
bubble_sort_provider_1.dart';
import 'package:sort_it_out/src/save_data/save_data_provider.dart';
import 'package:sort_it_out/src/score_system/score_system_provider.dart';

class BubbleSort1 extends StatefulWidget {
  const BubbleSort1({Key? key}) : super(key: key);

  @override
  State<StatefulWidget> createState() => _BubbleSort1State();
}

class _BubbleSort1State extends State<BubbleSort1> {
  @override
  Widget build(BuildContext context) {
    BubbleSortProvider1 _bubbleSortProvider =
        Provider.of<BubbleSortProvider1>(context);
    ScoreSystemProvider _scoreSystemProvider =
        Provider.of<ScoreSystemProvider>(context);
    BubbleSortMessageProvider _messageProvider =
        Provider.of<BubbleSortMessageProvider>(context);
    SaveDataProvider _saveProvider = Provider.of<SaveDataProvider>(context);
    return Scaffold(
        appBar: AppBar(
          title: const Text('Algoritmo 1 - Bubble Sort'),
          centerTitle: true,
        ),
        body: Column(
```

```
mainAxisAlignment: MainAxisAlignment.spaceBetween,
children: [
  Row(
    mainAxisAlignment: MainAxisAlignment.spaceEvenly,
    children: [
      Padding(
        padding: const EdgeInsets.all(8.0),
        child: TextButton(
            onPressed: () {
              {
                Future.delayed(const Duration(seconds: 1));
                _hintAsked(_scoreSystemProvider, _messageProvider);
              }
            },
            child: const Text('Pedir dica (-20 pontos)')),
      ),
      Padding(
        padding: const EdgeInsets.all(8.0),
        child: TextButton(
            onPressed: () {
              {
                if (_bubbleSortProvider
                    .checkCorrectMoveConditions()) {
                  _correctMove(_bubbleSortProvider,
                      _scoreSystemProvider, _messageProvider);
                } else {
                  _wrongMove(false, _bubbleSortProvider,
                      _scoreSystemProvider, _messageProvider);
                }
              }
            },
            child: const Text('Passar')),
      ),
      const SizedBox(height: 20),
      Padding(
        padding: const EdgeInsets.all(8.0),
```

```
            child:
                Text('Score: ' + _scoreSystemProvider.score.toString()),
            )
          ],
        ),
        ConstrainedBox(
          constraints: const BoxConstraints(
            maxHeight: 60,
            minWidth: 300,
          ),
          child: ReorderableListView(
              padding: const EdgeInsets.only(bottom: 10),
              shrinkWrap: true,
              scrollDirection: Axis.horizontal,
              children: _bubbleSortProvider.draggableList,
              onReorder: (oldIndex, newIndex) {
                _onReorder(oldIndex, newIndex, _bubbleSortProvider,
                    _scoreSystemProvider, _messageProvider);
                if (_bubbleSortProvider.isStageSolved) {
                  _onStageSolved(_bubbleSortProvider, _saveProvider);
                }
              }),
        ),
        Padding(
          padding: const EdgeInsets.all(24.0),
          child: Text(_messageProvider.currentMessage),
        ),
        const SizedBox(
          height: 60,
        )
      ],
    ));
}

void _onReorder(
    int oldIndex,
```

```
    int newIndex,
    BubbleSortProvider1 bubbleSortProvider,
    ScoreSystemProvider scoreSystemProvider,
    BubbleSortMessageProvider messageProvider) {
  setState(() {
    if (bubbleSortProvider.isPos0to1Move(oldIndex, newIndex)) {
      newIndex = newIndex - 1;
      final element = bubbleSortProvider.draggableList.removeAt(oldIndex);
      bubbleSortProvider.draggableList.insert(newIndex, element);
      if (bubbleSortProvider.checkCorrectMoveConditions()) {
        _correctMove(
            bubbleSortProvider, scoreSystemProvider, messageProvider);
      } else {
        _wrongMove(
            false, bubbleSortProvider, scoreSystemProvider, messageProvider);
      }
    } else if (bubbleSortProvider.isPos1to0Move(oldIndex, newIndex)) {
      final element = bubbleSortProvider.draggableList.removeAt(oldIndex);
      bubbleSortProvider.draggableList.insert(newIndex, element);
      if (bubbleSortProvider.checkCorrectMoveConditions()) {
        _correctMove(
            bubbleSortProvider, scoreSystemProvider, messageProvider);
      } else {
        _wrongMove(
            false, bubbleSortProvider, scoreSystemProvider, messageProvider);
      }
    } else {
      _wrongMove(
          false, bubbleSortProvider, scoreSystemProvider, messageProvider);
    }
  });
}

_onStageSolved(
    BubbleSortProvider1 bubbleSortProvider, SaveDataProvider saveProvider) {
  showDialog(
```

```
      context: context,
      builder: (BuildContext context) => AlertDialog(
          title: const Text('Sucesso!'),
          content: const Text('Parabéns! Você resolveu o desafio!'),
          actions: <Widget>[
            TextButton(
              onPressed: () {
                saveProvider.saveData!.bubbleSortSaveData.isStage1Complete =
                    true;
                saveProvider.saveStageCompletion();
                bubbleSortProvider.reset();
                Navigator.pop(context, 'OK');
              },
              child: const Text('OK'),
            ),
          ],
        ));
}

_hintAsked(ScoreSystemProvider scoreSystemProvider,
    BubbleSortMessageProvider messageProvider) {
  scoreSystemProvider.hintAsked();
  messageProvider.hintAsked();
}

_correctMove(
    BubbleSortProvider1 bubbleSortProvider,
    ScoreSystemProvider scoreSystemProvider,
    BubbleSortMessageProvider messageProvider) {
  bubbleSortProvider.correctMove();
  scoreSystemProvider.correctMove();
  messageProvider.correctMoveMessage();
}

_wrongMove(
    bool notHighlitedElementMoved,
```

```
    BubbleSortProvider1 bubbleSortProvider,
    ScoreSystemProvider scoreSystemProvider,
    BubbleSortMessageProvider messageProvider) {
  bubbleSortProvider.wrongMove();
  scoreSystemProvider.wrongMove();
  messageProvider.wrongMoveMessage(false);
 }
}
```

bubble_sort_provider_1.dart

```
import 'dart:developer';

import 'package:flutter/material.dart';
import 'package:sort_it_out/src/model/sortable_item.dart';

class BubbleSortProvider1 extends ChangeNotifier {
 late List<SortableItem> draggableList;
 late int _index1;
 late int _index2;
 bool isStageSolved = false;

 BubbleSortProvider1() {
  draggableList = _buildDraggableItemList(_generateListOfElements(10));
  _index1 = 0;
  _index2 = 1;
  isStageSolved = _isListFullyOrdered();
  select(_index1, _index2);
 }

 List<int> _generateListOfElements(int listSize) {
  List<int> listOfRandomNumbers =
      List.generate(listSize * 2, (index) => index);
  listOfRandomNumbers.shuffle();
  return listOfRandomNumbers.sublist(0, listSize);
 }
```

```
List<SortableItem> _buildDraggableItemList(List<int> randomNumbers) {
  List<SortableItem> draggableItemList = [];
  for (int number in randomNumbers) {
    SortableItem item = SortableItem(
        key: ValueKey(number), color: Colors.white, title: number.toString());
    draggableItemList.add(item);
  }
  return draggableItemList;
}

void correctMove() {
  //Change colors of old selected numbers.
  _unselect(_index1, _index2);
  _incrementIndexes();

  if (!isStageSolved) {
    if (_isSelectionOutOfBounds(_index2)) {
      select(0, 1);
      resetIndexes();
    } else {
      select(_index1, _index2);
    }
    log("index1: $_index1 e index2 $_index2");
    notifyListeners();
  }
}

void _unselect(int index1, int index2) {
  draggableList[index1] = SortableItem(
      title: draggableList[index1].title,
      color: Colors.white,
      key: draggableList[index1].key);
  draggableList[index2] = SortableItem(
      title: draggableList[index2].title,
      color: Colors.white,
```

```
      key: draggableList[index2].key);
}

void _incrementIndexes() {
  _index1++;
  _index2++;
  if (_isListFullyOrdered()) {
    isStageSolved = true;
    log('Stage solved!');
    notifyListeners();
  }
}

void select(int index1, int index2) {
  draggableList[index1] = SortableItem(
      title: draggableList[index1].title,
      color: Colors.green,
      key: draggableList[index1].key);
  draggableList[index2] = SortableItem(
      title: draggableList[index2].title,
      color: Colors.green,
      key: draggableList[index2].key);
}

void wrongMove() {
  draggableList[_index1] = SortableItem(
      title: draggableList[_index1].title,
      color: Colors.red,
      key: draggableList[_index1].key);
  draggableList[_index2] = SortableItem(
      title: draggableList[_index2].title,
      color: Colors.red,
      key: draggableList[_index2].key);
  notifyListeners();
}
```

```
bool checkCorrectMoveConditions() {
  return _isSelectionOrdered();
}

bool isPos0to1Move(int oldIndex, int newIndex) =>
    newIndex == oldIndex + 2 &&
    (oldIndex == _index1 && newIndex == _index2 + 1);

bool isPos1to0Move(int oldIndex, int newIndex) =>
    newIndex == oldIndex - 1 && (oldIndex == _index2 && newIndex == _index1);

bool _isSelectionOrdered() =>
    int.parse(draggableList[_index1].title) <
    int.parse(draggableList[_index2].title);

bool _isListFullyOrdered() {
  List<SortableItem> tempList = draggableList.toList();
  tempList.sort((a, b) => int.parse(a.title).compareTo(int.parse(b.title)));
  for (int i = 0; i < draggableList.length; i++) {
    int compareResult = int.parse(draggableList[i].title)
        .compareTo(int.parse(tempList[i].title));
    if (compareResult != 0) {
      log('Not ordered yet!');
      return false;
    }
  }
  return true;
}

bool _isSelectionOutOfBounds(int index) => _index2 == draggableList.length;

void resetIndexes() {
  _index1 = 0;
  _index2 = 1;
}
```

```
  void reset() {
    draggableList = _buildDraggableItemList(_generateListOfElements(10));
    _index1 = 0;
    _index2 = 1;
    isStageSolved = _isListFullyOrdered();
    select(_index1, _index2);
    notifyListeners();
  }
}
```

bubble_sort_2.dart

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'package:sort_it_out/src/game_modes/arcade/stages/bubble_sort/
bubble_sort_message_provider.dart';
import 'package:sort_it_out/src/game_modes/arcade/stages/bubble_sort/stage_2/
bubble_sort_provider_2.dart';
import 'package:sort_it_out/src/save_data/save_data_provider.dart';
import 'package:sort_it_out/src/score_system/score_system_provider.dart';

class BubbleSort2 extends StatefulWidget {
  const BubbleSort2({Key? key}) : super(key: key);

  @override
  State<StatefulWidget> createState() => _BubbleSort2State();
}

class _BubbleSort2State extends State<BubbleSort2> {
  @override
  Widget build(BuildContext context) {
    BubbleSortProvider2 _bubbleSortProvider =
        Provider.of<BubbleSortProvider2>(context);
    ScoreSystemProvider _scoreSystemProvider =
        Provider.of<ScoreSystemProvider>(context);
    BubbleSortMessageProvider _messageProvider =
```

```
    Provider.of<BubbleSortMessageProvider>(context);
SaveDataProvider _saveProvider = Provider.of<SaveDataProvider>(context);

return Scaffold(
    appBar: AppBar(
      title: const Text('Algoritmo 2 - Bubble Sort'),
      centerTitle: true,
    ),
    body: Column(
      mainAxisAlignment: MainAxisAlignment.spaceBetween,
      children: [
        Row(
          mainAxisAlignment: MainAxisAlignment.spaceEvenly,
          children: [
            Padding(
              padding: const EdgeInsets.all(8.0),
              child: TextButton(
                  onPressed: () {
                    {
                      if (_bubbleSortProvider
                          .checkCorrectMoveConditions()) {
                        _correctMove(_bubbleSortProvider,
                            _scoreSystemProvider, _messageProvider);
                      } else {
                        _wrongMove(false, _bubbleSortProvider,
                            _scoreSystemProvider, _messageProvider);
                      }
                    }
                  },
                  child: const Text('Passar')),
            ),
            const SizedBox(height: 20),
            Padding(
              padding: const EdgeInsets.all(8.0),
              child:
                  Text('Score: ' + _scoreSystemProvider.score.toString()),
```

```
            )
          ],
        ),
        ConstrainedBox(
          constraints: const BoxConstraints(
            maxHeight: 60,
            minWidth: 300,
          ),
          child: ReorderableListView(
              padding: const EdgeInsets.only(bottom: 10),
              shrinkWrap: true,
              scrollDirection: Axis.horizontal,
              children: _bubbleSortProvider.draggableList,
              onReorder: (oldIndex, newIndex) {
                _onReorder(oldIndex, newIndex, _bubbleSortProvider,
                    _scoreSystemProvider, _messageProvider);
                if (_bubbleSortProvider.isStageSolved) {
                  _onStageSolved(_bubbleSortProvider, _saveProvider);
                }
              }),
        ),
        Padding(
          padding: const EdgeInsets.all(24.0),
          child: Text(_messageProvider.currentMessage),
        ),
        const SizedBox(
          height: 60,
        )
      ],
    ));
}

void _onReorder(
    int oldIndex,
    int newIndex,
    BubbleSortProvider2 bubbleSortProvider,
```

```
    ScoreSystemProvider scoreSystemProvider,
    BubbleSortMessageProvider messageProvider) {
  setState(() {
    if (bubbleSortProvider.isPos0to1Move(oldIndex, newIndex)) {
      newIndex = newIndex - 1;
      final element = bubbleSortProvider.draggableList.removeAt(oldIndex);
      bubbleSortProvider.draggableList.insert(newIndex, element);
      if (bubbleSortProvider.checkCorrectMoveConditions()) {
        _correctMove(
            bubbleSortProvider, scoreSystemProvider, messageProvider);
      }
    } else if (bubbleSortProvider.isPos1to0Move(oldIndex, newIndex)) {
      final element = bubbleSortProvider.draggableList.removeAt(oldIndex);
      bubbleSortProvider.draggableList.insert(newIndex, element);
      if (bubbleSortProvider.checkCorrectMoveConditions()) {
        _correctMove(
            bubbleSortProvider, scoreSystemProvider, messageProvider);
      }
    } else {
      _wrongMove(
          false, bubbleSortProvider, scoreSystemProvider, messageProvider);
    }
  });
}

_onStageSolved(
    BubbleSortProvider2 bubbleSortProvider, SaveDataProvider saveProvider) {
  showDialog(
      context: context,
      builder: (BuildContext context) => AlertDialog(
          title: const Text('Sucesso!'),
          content: const Text('Parabéns! Você resolveu o desafio!'),
          actions: <Widget>[
            TextButton(
              onPressed: () {
                saveProvider.saveData!.bubbleSortSaveData.isStage2Complete =
```

```
                    true;
                saveProvider.saveStageCompletion();
                bubbleSortProvider.reset();
                Navigator.pop(context, 'OK');
              },
              child: const Text('OK'),
            ),
          ],
        ));
  }

  _correctMove(
      BubbleSortProvider2 bubbleSortProvider,
      ScoreSystemProvider scoreSystemProvider,
      BubbleSortMessageProvider messageProvider) {
    bubbleSortProvider.correctMove();
    scoreSystemProvider.correctMove();
    messageProvider.correctMoveMessage();
  }

  _wrongMove(
      bool notHighlitedElementMoved,
      BubbleSortProvider2 bubbleSortProvider,
      ScoreSystemProvider scoreSystemProvider,
      BubbleSortMessageProvider messageProvider) {
    bubbleSortProvider.wrongMove();
    scoreSystemProvider.wrongMove();
    messageProvider.wrongMoveMessage(false);
  }
}
```

bubble_sort_provider_2.dart

```
import 'dart:developer';
import 'package:flutter/material.dart';
import 'package:sort_it_out/src/model/sortable_item.dart';
```

```
class BubbleSortProvider2 extends ChangeNotifier {
  late List<SortableItem> draggableList;
  late int _index1;
  late int _index2;
  bool isStageSolved = false;

  BubbleSortProvider2() {
    draggableList = _buildDraggableItemList(_generateListOfElements(10));
    _index1 = 0;
    _index2 = 1;
    isStageSolved = _isListFullyOrdered();
    select(_index1, _index2);
  }

  List<int> _generateListOfElements(int listSize) {
    List<int> listOfRandomNumbers =
        List.generate(listSize * 2, (index) => index);
    listOfRandomNumbers.shuffle();
    return listOfRandomNumbers.sublist(0, listSize);
  }

  List<SortableItem> _buildDraggableItemList(List<int> randomNumbers) {
    List<SortableItem> draggableItemList = [];
    for (int number in randomNumbers) {
      SortableItem item = SortableItem(
          key: ValueKey(number), color: Colors.white, title: number.toString());
      draggableItemList.add(item);
    }
    return draggableItemList;
  }

  void correctMove() {
    //Change colors of old selected numbers.
    _unselect(_index1, _index2);
    _incrementIndexes();
```

```
  if (!isStageSolved) {
    if (_isSelectionOutOfBounds(_index2)) {
      select(0, 1);
      resetIndexes();
    } else {
      select(_index1, _index2);
    }
    log("index1: $_index1 e index2 $_index2");
    notifyListeners();
  }
}

void _unselect(int index1, int index2) {
  draggableList[index1] = SortableItem(
      title: draggableList[index1].title,
      color: Colors.white,
      key: draggableList[index1].key);
  draggableList[index2] = SortableItem(
      title: draggableList[index2].title,
      color: Colors.white,
      key: draggableList[index2].key);
}

void _incrementIndexes() {
  _index1++;
  _index2++;
  if (_isListFullyOrdered()) {
    isStageSolved = true;
    log('Stage solved!');
    notifyListeners();
  }
}

void select(int index1, int index2) {
  draggableList[index1] = SortableItem(
```

```
      title: draggableList[index1].title,
      color: Colors.green,
      key: draggableList[index1].key);
  draggableList[index2] = SortableItem(
      title: draggableList[index2].title,
      color: Colors.green,
      key: draggableList[index2].key);
}

void wrongMove() {
  draggableList[_index1] = SortableItem(
      title: draggableList[_index1].title,
      color: Colors.red,
      key: draggableList[_index1].key);
  draggableList[_index2] = SortableItem(
      title: draggableList[_index2].title,
      color: Colors.red,
      key: draggableList[_index2].key);
  notifyListeners();
}

bool checkCorrectMoveConditions() {
  return _isSelectionOrdered();
}

bool isPos0to1Move(int oldIndex, int newIndex) =>
    newIndex == oldIndex + 2 &&
    (oldIndex == _index1 && newIndex == _index2 + 1);

bool isPos1to0Move(int oldIndex, int newIndex) =>
    newIndex == oldIndex - 1 && (oldIndex == _index2 && newIndex == _index1);

bool _isSelectionOrdered() =>
    int.parse(draggableList[_index1].title) <
    int.parse(draggableList[_index2].title);
```

```
  bool _isListFullyOrdered() {
    List<SortableItem> tempList = draggableList.toList();
    tempList.sort((a, b) => int.parse(a.title).compareTo(int.parse(b.title)));
    for (int i = 0; i < draggableList.length; i++) {
      int compareResult = int.parse(draggableList[i].title)
          .compareTo(int.parse(tempList[i].title));
      if (compareResult != 0) {
        log('Not ordered yet!');
        return false;
      }
    }
    return true;
  }

  bool _isSelectionOutOfBounds(int index) => _index2 == draggableList.length;

  void resetIndexes() {
    _index1 = 0;
    _index2 = 1;
  }

  void reset() {
    draggableList = _buildDraggableItemList(_generateListOfElements(10));
    _index1 = 0;
    _index2 = 1;
    isStageSolved = _isListFullyOrdered();
    select(_index1, _index2);
    notifyListeners();
  }
}
```

bubble_sort_3.dart

```
import 'dart:developer';
import 'package:flutter/material.dart';
import 'package:sort_it_out/src/model/sortable_item.dart';
```

```
class BubbleSortProvider2 extends ChangeNotifier {
  late List<SortableItem> draggableList;
  late int _index1;
  late int _index2;
  bool isStageSolved = false;

  BubbleSortProvider2() {
    draggableList = _buildDraggableItemList(_generateListOfElements(10));
    _index1 = 0;
    _index2 = 1;
    isStageSolved = _isListFullyOrdered();
    select(_index1, _index2);
  }

  List<int> _generateListOfElements(int listSize) {
    List<int> listOfRandomNumbers =
        List.generate(listSize * 2, (index) => index);
    listOfRandomNumbers.shuffle();
    return listOfRandomNumbers.sublist(0, listSize);
  }

  List<SortableItem> _buildDraggableItemList(List<int> randomNumbers) {
    List<SortableItem> draggableItemList = [];
    for (int number in randomNumbers) {
      SortableItem item = SortableItem(
          key: ValueKey(number), color: Colors.white, title: number.toString());
      draggableItemList.add(item);
    }
    return draggableItemList;
  }

  void correctMove() {
    //Change colors of old selected numbers.
    _unselect(_index1, _index2);
    _incrementIndexes();
```

```
  if (!isStageSolved) {
    if (_isSelectionOutOfBounds(_index2)) {
      select(0, 1);
      resetIndexes();
    } else {
      select(_index1, _index2);
    }
    log("index1: $_index1 e index2 $_index2");
    notifyListeners();
  }
}

void _unselect(int index1, int index2) {
  draggableList[index1] = SortableItem(
      title: draggableList[index1].title,
      color: Colors.white,
      key: draggableList[index1].key);
  draggableList[index2] = SortableItem(
      title: draggableList[index2].title,
      color: Colors.white,
      key: draggableList[index2].key);
}

void _incrementIndexes() {
  _index1++;
  _index2++;
  if (_isListFullyOrdered()) {
    isStageSolved = true;
    log('Stage solved!');
    notifyListeners();
  }
}

void select(int index1, int index2) {
  draggableList[index1] = SortableItem(
```

```
      title: draggableList[index1].title,
      color: Colors.green,
      key: draggableList[index1].key);
  draggableList[index2] = SortableItem(
      title: draggableList[index2].title,
      color: Colors.green,
      key: draggableList[index2].key);
}

void wrongMove() {
  draggableList[_index1] = SortableItem(
      title: draggableList[_index1].title,
      color: Colors.red,
      key: draggableList[_index1].key);
  draggableList[_index2] = SortableItem(
      title: draggableList[_index2].title,
      color: Colors.red,
      key: draggableList[_index2].key);
  notifyListeners();
}

bool checkCorrectMoveConditions() {
  return _isSelectionOrdered();
}

bool isPos0to1Move(int oldIndex, int newIndex) =>
    newIndex == oldIndex + 2 &&
    (oldIndex == _index1 && newIndex == _index2 + 1);

bool isPos1to0Move(int oldIndex, int newIndex) =>
    newIndex == oldIndex - 1 && (oldIndex == _index2 && newIndex == _index1);

bool _isSelectionOrdered() =>
    int.parse(draggableList[_index1].title) <
    int.parse(draggableList[_index2].title);
```

```
  bool _isListFullyOrdered() {
    List<SortableItem> tempList = draggableList.toList();
    tempList.sort((a, b) => int.parse(a.title).compareTo(int.parse(b.title)));
    for (int i = 0; i < draggableList.length; i++) {
      int compareResult = int.parse(draggableList[i].title)
          .compareTo(int.parse(tempList[i].title));
      if (compareResult != 0) {
        log('Not ordered yet!');
        return false;
      }
    }
    return true;
  }

  bool _isSelectionOutOfBounds(int index) => _index2 == draggableList.length;

  void resetIndexes() {
    _index1 = 0;
    _index2 = 1;
  }

  void reset() {
    draggableList = _buildDraggableItemList(_generateListOfElements(10));
    _index1 = 0;
    _index2 = 1;
    isStageSolved = _isListFullyOrdered();
    select(_index1, _index2);
    notifyListeners();
  }
}
```

bubble_sort_provider_3

```
import 'dart:developer';
import 'package:flutter/material.dart';
import 'package:sort_it_out/src/model/sortable_item.dart';
```

```
class BubbleSortProvider3 extends ChangeNotifier {
  late List<SortableItem> draggableList;
  late int _index1;
  late int _index2;
  bool isStageSolved = false;

  BubbleSortProvider3() {
    draggableList = _buildDraggableItemList(_generateListOfElements(10));
    _index1 = 0;
    _index2 = 1;
    isStageSolved = _isListFullyOrdered();
    select(_index1, _index2);
  }

  List<int> _generateListOfElements(int listSize) {
    List<int> listOfRandomNumbers =
        List.generate(listSize * 2, (index) => index);
    listOfRandomNumbers.shuffle();
    return listOfRandomNumbers.sublist(0, listSize);
  }

  List<SortableItem> _buildDraggableItemList(List<int> randomNumbers) {
    List<SortableItem> draggableItemList = [];
    for (int number in randomNumbers) {
      SortableItem item = SortableItem(
          key: ValueKey(number), color: Colors.white, title: number.toString());
      draggableItemList.add(item);
    }
    return draggableItemList;
  }

  void correctMove() {
    //Change colors of old selected numbers.
    _unselect(_index1, _index2);
    _incrementIndexes();
```

```
  if (!isStageSolved) {
    if (_isSelectionOutOfBounds(_index2)) {
      select(0, 1);
      resetIndexes();
    } else {
      select(_index1, _index2);
    }
    log("index1: $_index1 e index2 $_index2");
    notifyListeners();
  }
}

void _unselect(int index1, int index2) {
  draggableList[index1] = SortableItem(
      title: draggableList[index1].title,
      color: Colors.white,
      key: draggableList[index1].key);
  draggableList[index2] = SortableItem(
      title: draggableList[index2].title,
      color: Colors.white,
      key: draggableList[index2].key);
}

void _incrementIndexes() {
  _index1++;
  _index2++;
  if (_isListFullyOrdered()) {
    isStageSolved = true;
    log('Stage solved!');
    notifyListeners();
  }
}

void select(int index1, int index2) {
  draggableList[index1] = SortableItem(
```

```
      title: draggableList[index1].title,
      color: Colors.green,
      key: draggableList[index1].key);
  draggableList[index2] = SortableItem(
      title: draggableList[index2].title,
      color: Colors.green,
      key: draggableList[index2].key);
}

void wrongMove() {
  draggableList[_index1] = SortableItem(
      title: draggableList[_index1].title,
      color: Colors.red,
      key: draggableList[_index1].key);
  draggableList[_index2] = SortableItem(
      title: draggableList[_index2].title,
      color: Colors.red,
      key: draggableList[_index2].key);
  notifyListeners();
}

bool checkCorrectMoveConditions() {
  return _isSelectionOrdered();
}

bool isPos0to1Move(int oldIndex, int newIndex) =>
    newIndex == oldIndex + 2 &&
    (oldIndex == _index1 && newIndex == _index2 + 1);

bool isPos1to0Move(int oldIndex, int newIndex) =>
    newIndex == oldIndex - 1 && (oldIndex == _index2 && newIndex == _index1);

bool _isSelectionOrdered() =>
    int.parse(draggableList[_index1].title) <
    int.parse(draggableList[_index2].title);
```

```
  bool _isListFullyOrdered() {
    List<SortableItem> tempList = draggableList.toList();
    tempList.sort((a, b) => int.parse(a.title).compareTo(int.parse(b.title)));
    for (int i = 0; i < draggableList.length; i++) {
      int compareResult = int.parse(draggableList[i].title)
          .compareTo(int.parse(tempList[i].title));
      if (compareResult != 0) {
        log('Not ordered yet!');
        return false;
      }
    }
    return true;
  }

  bool _isSelectionOutOfBounds(int index) => _index2 == draggableList.length;

  void resetIndexes() {
    _index1 = 0;
    _index2 = 1;
  }

  void reset() {
    draggableList = _buildDraggableItemList(_generateListOfElements(10));
    _index1 = 0;
    _index2 = 1;
    isStageSolved = _isListFullyOrdered();
    select(_index1, _index2);
    notifyListeners();
  }
}
```

bubble_sort_message_provider

```
import 'package:flutter/cupertino.dart';

class BubbleSortMessageProvider extends ChangeNotifier {
```

```
  late String currentMessage;

  late List<String> _hints;

  BubbleSortMessageProvider() {
    currentMessage = '';
    _hints = [
      'Lembre-se que o elemento destacado à esquerda deve sempre ser menor que o
elemento à direita'
    ];
  }

  void wrongMoveMessage(bool notHighlitedElementMoved) {
    if (notHighlitedElementMoved) {
      currentMessage = 'Elementos não destacados não devem ser movidos!';
      notifyListeners();
    } else {
      currentMessage = 'Movimento incorreto!';
      notifyListeners();
    }
  }

  void correctMoveMessage() {
    currentMessage = 'Bom trabalho!';
    notifyListeners();
  }

  void hintAsked() {
    currentMessage = _getRandomHint();
    notifyListeners();
  }

  String _getRandomHint() {
    List<int> listOfRandomNumbers =
        List.generate(_hints.length, (index) => index);
    listOfRandomNumbers.shuffle();
```

```
    return _hints[listOfRandomNumbers.first];
  }
}
```

bubble_sort_message_provider.dart

```
import 'package:flutter/cupertino.dart';

class BubbleSortMessageProvider extends ChangeNotifier {
  late String currentMessage;

  late List<String> _hints;

  BubbleSortMessageProvider() {
    currentMessage = '';
    _hints = [
      'Lembre-se que o elemento destacado à esquerda deve sempre ser menor que o
elemento à direita'
    ];
  }

  void wrongMoveMessage(bool notHighlitedElementMoved) {
    if (notHighlitedElementMoved) {
      currentMessage = 'Elementos não destacados não devem ser movidos!';
      notifyListeners();
    } else {
      currentMessage = 'Movimento incorreto!';
      notifyListeners();
    }
  }

  void correctMoveMessage() {
    currentMessage = 'Bom trabalho!';
    notifyListeners();
  }
```

```
  void hintAsked() {
    currentMessage = _getRandomHint();
    notifyListeners();
  }

  String _getRandomHint() {
    List<int> listOfRandomNumbers =
        List.generate(_hints.length, (index) => index);
    listOfRandomNumbers.shuffle();
    return _hints[listOfRandomNumbers.first];
  }
}
```

merge_sort_1.dart

```
import 'dart:developer';

import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'package:sort_it_out/src/game_modes/arcade/stages/merge_sort/
merge_sort_message_provider.dart';
import 'package:sort_it_out/src/game_modes/arcade/stages/merge_sort/stage_1/
merge_sort_provider_1.dart';
import 'package:sort_it_out/src/model/sortable_item.dart';
import 'package:sort_it_out/src/save_data/save_data_provider.dart';
import 'package:sort_it_out/src/score_system/score_system_provider.dart';

class MergeSort1 extends StatefulWidget {
  const MergeSort1({Key? key}) : super(key: key);

  @override
  State<StatefulWidget> createState() => _MergeSort1State();
}

class _MergeSort1State extends State<MergeSort1> {
  @override
```

```
Widget build(BuildContext context) {
  MergeSortProvider1 _mergeSortProvider =
      Provider.of<MergeSortProvider1>(context);
  ScoreSystemProvider _scoreSystemProvider =
      Provider.of<ScoreSystemProvider>(context);
  MergeSortMessageProvider _messageProvider =
      Provider.of<MergeSortMessageProvider>(context);
  SaveDataProvider _saveProvider = Provider.of<SaveDataProvider>(context);
  return Scaffold(
    appBar: AppBar(
      title: const Text('Algoritmo 1 - Merge Sort'),
      centerTitle: true,
    ),
    body: Padding(
      padding: const EdgeInsets.only(left: 8.0),
      child: Row(
        mainAxisAlignment: MainAxisAlignment.spaceBetween,
        children: [
          SizedBox(
            width: 270,
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.center,
              mainAxisAlignment: MainAxisAlignment.start,
              children: [
                Column(
                  children: [
                    const SizedBox(height: 20),
                    Padding(
                      padding: const EdgeInsets.all(8.0),
                      child: Text(
                          'Score: ' + _scoreSystemProvider.score.toString()),
                    )
                  ],
                ),
                Padding(
                  padding: const EdgeInsets.all(16.0),
```

```
        child: TextButton(
          onPressed: () {
            {
              Future.delayed(const Duration(seconds: 1));
              _hintAsked(_scoreSystemProvider, _messageProvider);
            }
          },
          child: const Text('Pedir dica (-20 pontos)')),
      ),
      SizedBox(
        width: 270,
        child: Padding(
          padding: const EdgeInsets.only(
              left: 16.0, right: 8.0, top: 8.0, bottom: 8.0),
          child: Text(
            _messageProvider.currentMessage,
            softWrap: true,
          ),
        ),
      ),
    ],
  ),
),
SingleChildScrollView(
  child: SizedBox(
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        ConstrainedBox(
          constraints: const BoxConstraints(
            maxHeight: 60,
            minWidth: 150,
          ),
          child: ListView(
              padding: const EdgeInsets.only(bottom: 10),
              shrinkWrap: true,
```

```
      scrollDirection: Axis.horizontal,
      children: _mergeSortProvider.getInitialList()),
),
for (List<List<SortableItem>> step
    in _mergeSortProvider.steps)
  Column(
    children: [
      Padding(
        padding: const EdgeInsets.only(bottom: 20),
        child: Column(
          children: [
            Row(
              crossAxisAlignment: CrossAxisAlignment.center,
              mainAxisAlignment: MainAxisAlignment.center,
              children: [
                for (List<SortableItem> internalList
                    in step)
                  IconButton(
                      onPressed: () => _selectInternalList(
                          internalList,
                          _mergeSortProvider,
                          _messageProvider,
                          _saveProvider,
                          _scoreSystemProvider),
                      icon: const Icon(
                          Icons.arrow_downward_sharp)),
                const SizedBox(
                  width: 36,
                )
              ],
            ),
            Row(
              mainAxisAlignment: MainAxisAlignment.center,
              children: _buildWholeStep(step),
            ),
          ],
```

```
                    ),
                  ),
                ],
              ),
            ],
          ),
        ),
      ),
      const SizedBox(
        width: 150,
      )
    ],
  ),
 ),
);
}

_buildWholeStep(List<List<SortableItem>> step) {
 List<Widget> list = [];
 for (int i = 0; i < step.length; i++) {
  list.add(Padding(
   padding: i == step.length - 1
     ? const EdgeInsets.only(right: 20)
     : EdgeInsets.zero,
   child: SizedBox(
    width: step[i].length * 40,
    height: 40,
    child: ListView(
     scrollDirection: Axis.horizontal,
     children: step[i],
    ),
   ),
  ));
 }
 return list;
}
```

```
_hintAsked(ScoreSystemProvider scoreSystemProvider,
    MergeSortMessageProvider messageProvider) {
  scoreSystemProvider.hintAsked();
  messageProvider.hintAsked();
}

_selectInternalList(
  List<SortableItem> internalList,
  MergeSortProvider1 mergeSortProvider,
  MergeSortMessageProvider messageProvider,
  SaveDataProvider saveDataProvider,
  ScoreSystemProvider scoreProvider,
) {
  bool result = mergeSortProvider.checkInternalListSelection(internalList);

  result
      ? _correctSelection(internalList, messageProvider, mergeSortProvider,
          saveDataProvider, scoreProvider)
      : _incorrectSelection(
          messageProvider, mergeSortProvider, scoreProvider);
  log('selected internalList: ${internalList.toString()} is it correct? [$result]');
}

_correctSelection(
  List<SortableItem> internalList,
  MergeSortMessageProvider messageProvider,
  MergeSortProvider1 mergeSortProvider,
  SaveDataProvider saveDataProvider,
  ScoreSystemProvider scoreProvider,
) {
  messageProvider.correctMoveMessage();
  scoreProvider.correctMove();
  mergeSortProvider.select(internalList);
  if (mergeSortProvider.isStageSolved(internalList)) {
    _onStageSolved(mergeSortProvider, saveDataProvider);
```

```
    }
  }

  _incorrectSelection(MergeSortMessageProvider messageProvider,
      MergeSortProvider1 mergeSortProvider, ScoreSystemProvider scoreProvider) {
    scoreProvider.wrongMove();
    messageProvider.wrongMoveMessage();
  }

  _onStageSolved(
      MergeSortProvider1 mergeSortProvider, SaveDataProvider saveProvider) {
    showDialog(
        context: context,
        builder: (BuildContext context) => AlertDialog(
              title: const Text('Sucesso!'),
              content: const Text('Parabéns! Você resolveu o desafio!'),
              actions: <Widget>[
                TextButton(
                  onPressed: () {
                    saveProvider.saveData!.mergeSortSaveData.isStage1Complete =
                        true;
                    saveProvider.saveStageCompletion();
                    mergeSortProvider.reset();
                    Navigator.pop(context, 'OK');
                  },
                  child: const Text('OK'),
                ),
              ],
            ));
  }
}
```

merge_sort_provider_1.dart

```
import 'dart:developer';
```

```
import 'package:flutter/foundation.dart';
import 'package:flutter/material.dart';
import 'package:sort_it_out/src/model/sortable_item.dart';
import 'package:collection/collection.dart';

class MergeSortProvider1 extends ChangeNotifier {
  late List<SortableItem> _initialList;
  late List<List<List<SortableItem>>> steps;
  late List<SortableItem> _finalList;
  final int _numberOfElementsToSort = 4;
  late List<List<SortableItem>> _selectedInternalLists;
  late List<List<SortableItem>> _correctOrderOfInternalListsSelection;

  MergeSortProvider1() {
    steps = [];
    _selectedInternalLists = [];
    _correctOrderOfInternalListsSelection = [];
    _createChallengeList(_numberOfElementsToSort);
    _generateSteps();
    _generateMergeSteps();
    _generateCorrectOrderOfInternalListsSelection();
  }

  List<int> _generateListOfElements(int listSize) {
    List<int> listOfRandomNumbers =
        List.generate(listSize * 2, (index) => index);
    listOfRandomNumbers.shuffle();
    List<int> returnList = listOfRandomNumbers.sublist(0, listSize);

    while (_isListOrdered(returnList)) {
      listOfRandomNumbers.shuffle();
      returnList = listOfRandomNumbers.sublist(0, listSize);
    }

    return returnList;
  }
```

```
List<SortableItem> _buildDraggableItemList(List<int> randomNumbers) {
  List<SortableItem> draggableItemList = [];
  for (int number in randomNumbers) {
    SortableItem item = SortableItem(
      key: ValueKey(number),
      color: Colors.white,
      title: number.toString(),
      height: 20.0,
      width: 30.0,
      fontSize: 15,
    );
    draggableItemList.add(item);
  }
  return draggableItemList;
}

bool checkInternalListSelection(List<SortableItem> internalList) {
  bool isCorrect = _isCorrectlySelected(internalList);
  if (isCorrect) {
    _selectedInternalLists.add(internalList);
  }
  return isCorrect;
}

_createChallengeList(int numberOfElements) {
  _initialList =
      _buildDraggableItemList(_generateListOfElements(numberOfElements));
  log("Lista inicial: $_initialList");
}

_generateSteps() {
  _splitSortableItemList([_initialList]);

  while (!_isEveryInternalListOneElement(steps.last)) {
    _splitSortableItemList(steps.last);
```

```
  }

  log("steps: " + steps.toString());
}

_generateMergeSteps() {
  List<List<List<SortableItem>>> splitSteps = steps;

  while (steps.last.length > 1) {
    List<List<SortableItem>> newStep = [];
    if (splitSteps.last.length % 2 == 0) {
      int mergeTimes = (splitSteps.last.length / 2).round();
      int index = 0;

      while (mergeTimes > 0) {
        List<SortableItem> list = List.from(splitSteps.last[index])
          ..addAll(splitSteps.last[index + 1]);
        list.sort((a, b) {
          if (int.parse(a.title) > int.parse(b.title)) {
            return 1;
          } else {
            if (int.parse(a.title) < int.parse(b.title)) {
              return -1;
            }
            return 0;
          }
        });
        newStep.add(list);
        mergeTimes--;
        index += 2;
      }
    }
    steps.add(newStep);
  }
  log('Steps after generating Merge steps: ' + steps.toString());
}
```

```
List<SortableItem> getInitialList() => _initialList;

List<SortableItem> getFinalList() => _finalList;

void _splitSortableItemList(List<List<SortableItem>> sortableItemList) {
  List<List<SortableItem>> newList = [];
  for (List<SortableItem> oldList in sortableItemList) {
    int listSize = oldList.length;
    if (listSize > 1) {
      if (listSize.isEven) {
        newList.add(oldList.sublist(0, (listSize / 2).floor()));
        newList.add(oldList.sublist((listSize / 2).floor()));
      } else {
        newList.add(oldList.sublist(0, (listSize / 2).floor() + 1));
        newList.add(oldList.sublist((listSize / 2).floor() + 1));
      }
    } else if (listSize == 1) {
      newList.add(oldList);
    }
  }
  steps.add(newList);
}

bool _isEveryInternalListOneElement(List<List<SortableItem>> externalList) {
  int result = 0;
  for (List<SortableItem> internalList in externalList) {
    if (internalList.length == 1) {
      result++;
    }
  }
  return result == externalList.length;
}

bool _isListOrdered(List<int> list) {
  return list.isSorted((a, b) => a.compareTo(b));
```

```
}

void _generateCorrectOrderOfInternalListsSelection() {
  _correctOrderOfInternalListsSelection.addAll([
    steps[0][0],
    steps[1][0],
    steps[1][1],
    steps[2][0],
    steps[0][1],
    steps[1][2],
    steps[1][3],
    steps[2][1],
    steps[3][0]
  ]);

  log('correct order of internal lists: ' +
      _correctOrderOfInternalListsSelection.toString());
}

bool _isCorrectlySelected(List<SortableItem> internalList) {
  if (_correctOrderOfInternalListsSelection.contains(internalList)) {
    if (_correctOrderOfInternalListsSelection[
            _selectedInternalLists.length] ==
        internalList) {
      return true;
    }
  }
  return false;
}

reset() {
  steps = [];
  _selectedInternalLists = [];
  _correctOrderOfInternalListsSelection = [];
  _createChallengeList(_numberOfElementsToSort);
  _generateSteps();
```

```
  _generateMergeSteps();
  _generateCorrectOrderOfInternalListsSelection();
}

bool isStageSolved(List<SortableItem> currentInternalList) {
  if (currentInternalList.length != steps.last.last.length) {
    return false;
  }
  int index = 0;
  for (SortableItem item in steps.last.last) {
    if (item.title != currentInternalList[index].title) {
      return false;
    }
    index++;
  }
  return true;
}

int _getLevelOfInternalList(List<SortableItem> selectedInternalList) {
  int outerIndex = 0;
  for (List<List<SortableItem>> step in steps) {
    for (List<SortableItem> internalList in step) {
      if (listEquals(internalList, selectedInternalList)) {
        return outerIndex;
      }
    }
    outerIndex++;
  }

  return -1;
}

void select(List<SortableItem> selectedInternalList) {
  int level = _getLevelOfInternalList(selectedInternalList);

  int index = 0;
```

```
    List<SortableItem> newInternalList = [];
    for (List<SortableItem> internalList in steps[level]) {
      if (listEquals(internalList, selectedInternalList)) {
        List<SortableItem> newInternalList = [];
        for (SortableItem item in steps[level][index]) {
          newInternalList.add(SortableItem(
              title: item.title,
              color: Colors.green,
              key: item.key,
              height: 20.0,
              width: 30.0,
              fontSize: 15));
        }

        steps[level][index] = newInternalList;
      }

      index++;
    }

    notifyListeners();
  }
}
```

merge_sort_2.dart

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'package:sort_it_out/src/game_modes/arcade/stages/merge_sort/
merge_sort_message_provider.dart';
import 'package:sort_it_out/src/game_modes/arcade/stages/merge_sort/stage_2/
merge_sort_provider_2.dart';
import 'package:sort_it_out/src/model/input_item.dart';
import 'package:sort_it_out/src/save_data/save_data_provider.dart';
import 'package:sort_it_out/src/score_system/score_system_provider.dart';
```

```
class MergeSort2 extends StatefulWidget {
  const MergeSort2({Key? key}) : super(key: key);

  @override
  State<StatefulWidget> createState() => _MergeSort2State();
}

class _MergeSort2State extends State<MergeSort2> {
  @override
  Widget build(BuildContext context) {
    MergeSortProvider2 _mergeSortProvider =
        Provider.of<MergeSortProvider2>(context);
    ScoreSystemProvider _scoreSystemProvider =
        Provider.of<ScoreSystemProvider>(context);
    MergeSortMessageProvider _messageProvider =
        Provider.of<MergeSortMessageProvider>(context);
    SaveDataProvider _saveProvider = Provider.of<SaveDataProvider>(context);
    return Scaffold(
      appBar: AppBar(
        title: const Text('Algoritmo 2 - Merge Sort'),
        centerTitle: true,
      ),
      body: Padding(
        padding: const EdgeInsets.only(left: 8.0),
        child: Row(
          mainAxisAlignment: MainAxisAlignment.spaceBetween,
          children: [
            Column(
              crossAxisAlignment: CrossAxisAlignment.center,
              mainAxisAlignment: MainAxisAlignment.start,
              children: [
                Column(
                  children: [
                    const SizedBox(height: 20),
                    Padding(
                      padding: const EdgeInsets.all(8.0),
```

```
          child: Text(
              'Score: ' + _scoreSystemProvider.score.toString()),
        )
      ],
    ),
    Padding(
      padding: const EdgeInsets.all(16.0),
      child: TextButton(
          onPressed: () {
            {
              Future.delayed(const Duration(seconds: 1));
              _hintAsked(_scoreSystemProvider, _messageProvider);
            }
          },
          child: const Text('Pedir dica (-20 pontos)')),
    ),
    SizedBox(
      width: 270,
      child: Text(
        _messageProvider.currentMessage,
        softWrap: true,
      ),
    ),
  ],
),
SingleChildScrollView(
  child: SizedBox(
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        ConstrainedBox(
          constraints: const BoxConstraints(
            maxHeight: 60,
            minWidth: 150,
          ),
          child: ListView(
```

```
      padding: const EdgeInsets.only(bottom: 10),
      shrinkWrap: true,
      scrollDirection: Axis.horizontal,
      children: _mergeSortProvider.getInitialList()),
),
for (List<List<InputItem>> step
    in _mergeSortProvider.inputs)
  Column(
    children: [
      Padding(
        padding: const EdgeInsets.only(bottom: 20),
        child: Column(
          children: [
            Row(children: [
              for (List<InputItem> internalList in step)
                step.last != internalList
                    ? Row(
                        children: const [
                          Icon(Icons.arrow_downward_sharp),
                          SizedBox(
                            width: 36,
                          )
                        ],
                      )
                    : const Icon(
                        Icons.arrow_downward_sharp),
              const SizedBox(
                width: 30,
              )
            ]),
            Row(
              mainAxisAlignment: MainAxisAlignment.center,
              children: _buildWholeStep(step),
            ),
          ],
        ),
```

```
                    ),
                  ],
                ),
              ],
            ),
          ),
        ),
        Column(
          children: [
            Padding(
              padding: const EdgeInsets.all(16.0),
              child: TextButton(
                  onPressed: () {
                    {
                      Future.delayed(const Duration(seconds: 1));
                      _submitInputValues(_mergeSortProvider, _saveProvider,
                          _scoreSystemProvider, _messageProvider);
                    }
                  },
                  child: const Text('Submeter')),
            ),
          ],
        ),
        const SizedBox(
          width: 16,
        )
      ],
    ),
  ),
 );
}

_buildWholeStep(List<List<InputItem>> step) {
  List<Widget> list = [];

  for (int i = 0; i < step.length; i++) {
```

```
    list.add(Padding(
      padding: i == step.length - 1
          ? const EdgeInsets.only(right: 20)
          : EdgeInsets.zero,
      child: Row(
        children: [
          SizedBox(
            width: step[i].length * 40,
            height: 40,
            child: ListView(
              scrollDirection: Axis.horizontal,
              children: step[i],
            ),
          ),
          const SizedBox(
            width: 10,
          )
        ],
      ),
    ));
  }
  return list;
}

_hintAsked(ScoreSystemProvider scoreSystemProvider,
    MergeSortMessageProvider messageProvider) {
  scoreSystemProvider.hintAsked();
  messageProvider.hintAsked();
}

_submitInputValues(
  MergeSortProvider2 mergeSortProvider,
  SaveDataProvider saveDataProvider,
  ScoreSystemProvider scoreProvider,
  MergeSortMessageProvider messageProvider,
) {
```

```
  bool isSubmissionCorrect = mergeSortProvider.checkInputs();
  if (isSubmissionCorrect) {
    _correctInputs(
        messageProvider, mergeSortProvider, saveDataProvider, scoreProvider);
  } else {
    if (_areThereEmptyInputs(mergeSortProvider)) {
      _emptyInputs(messageProvider, mergeSortProvider);
    } else {
      _incorrectInputs(messageProvider, mergeSortProvider, scoreProvider);
    }
  }
}

bool _areThereEmptyInputs(MergeSortProvider2 mergeSortProvider) {
  return mergeSortProvider.areThereEmptyInputs();
}

_correctInputs(
    MergeSortMessageProvider messageProvider,
    MergeSortProvider2 mergeSortProvider,
    SaveDataProvider saveDataProvider,
    ScoreSystemProvider scoreProvider) {
  messageProvider.correctMoveMessage();
  scoreProvider.correctMove();
  _onStageSolved(mergeSortProvider, saveDataProvider);
}

_incorrectInputs(MergeSortMessageProvider messageProvider,
    MergeSortProvider2 mergeSortProvider, ScoreSystemProvider scoreProvider) {
  scoreProvider.wrongMove();
  messageProvider.wrongMoveMessageInputs();
  mergeSortProvider.highlightWrongInputs();
}

_emptyInputs(MergeSortMessageProvider messageProvider,
    MergeSortProvider2 mergeSortProvider) {
```

```
    // --- perguntar se deve fazer o usuario perder pontos
    messageProvider.emptyInputsErrorMessage();
  }

  _onStageSolved(
      MergeSortProvider2 mergeSortProvider, SaveDataProvider saveProvider) {
    showDialog(
        context: context,
        builder: (BuildContext context) => AlertDialog(
              title: const Text('Sucesso!'),
              content: const Text('Parabéns! Você resolveu o desafio!'),
              actions: <Widget>[
                TextButton(
                  onPressed: () {
                    saveProvider.saveData!.mergeSortSaveData.isStage2Complete =
                        true;
                    saveProvider.saveStageCompletion();
                    mergeSortProvider.reset();
                    Navigator.pop(context, 'OK');
                  },
                  child: const Text('OK'),
                ),
              ],
            ));
  }
}
```

merge_sort_provider_2.dart

```
import 'dart:developer';
import 'package:flutter/foundation.dart';
import 'package:flutter/material.dart';
import 'package:sort_it_out/src/model/input_item.dart';
import 'package:sort_it_out/src/model/sortable_item.dart';
import 'package:collection/collection.dart';
```

```
class MergeSortProvider2 extends ChangeNotifier {
  late List<SortableItem> _initialList;
  late List<List<List<SortableItem>>> _steps;
  late List<SortableItem> _finalList;
  bool isStageSolved = false;
  final int _numberOfElementsToSort = 4;
  late List<int> _correctOrderOfInputs;
  late List<List<List<InputItem>>> inputs;
  late List<String> _currentInputValues;

  MergeSortProvider2() {
    _steps = [];
    _correctOrderOfInputs = [];
    inputs = [];
    _currentInputValues = [];
    _createChallengeList(_numberOfElementsToSort);
    _generateSteps();
    _generateMergeSteps();
    _generateCorrectOrderOfInputs();
    _stepsToInputs();
    _assignOnChangeFunctionToInputs(true);
  }

  List<int> _generateListOfElements(int listSize) {
    List<int> listOfRandomNumbers =
        List.generate(listSize * 2, (index) => index);
    listOfRandomNumbers.shuffle();
    List<int> returnList = listOfRandomNumbers.sublist(0, listSize);

    while (_isListOrdered(returnList)) {
      listOfRandomNumbers.shuffle();
      returnList = listOfRandomNumbers.sublist(0, listSize);
    }

    return returnList;
  }
```

```
List<SortableItem> _buildDraggableItemList(List<int> randomNumbers) {
  List<SortableItem> draggableItemList = [];
  for (int number in randomNumbers) {
    SortableItem item = SortableItem(
      key: ValueKey(number),
      color: Colors.white,
      title: number.toString(),
      height: 20.0,
      width: 30.0,
      fontSize: 15,
    );
    draggableItemList.add(item);
  }
  return draggableItemList;
}

highlightWrongInputs() {
  int index = 0;
  List<int> wrongInputPositions = [];
  for (String element in _currentInputValues) {
    if (int.parse(element) != _correctOrderOfInputs[index]) {
      wrongInputPositions.add(index);
    }
    index++;
  }

  log('These elements are wrong: ${wrongInputPositions.toString()}');

  _selectWrongInputs(wrongInputPositions);
}

bool areThereEmptyInputs() {
  List<int> emptyInputsPositions = [];
  int index = 0;
  for (String inputValue in _currentInputValues) {
```

```
    if (inputValue.isEmpty) {
      emptyInputsPositions.add(index);
    }
    index++;
  }

  if (emptyInputsPositions.isEmpty) {
    return false;
  } else {
    _selectWrongInputs(emptyInputsPositions);
    return true;
  }
}

bool checkInputs() {
  List<int> currentValuesToIntList = [];
  for (String inputValue in _currentInputValues) {
    currentValuesToIntList.add(int.tryParse(inputValue) ?? -1);
  }
  bool isCurrentSubmissionCorrect =
      listEquals(currentValuesToIntList, _correctOrderOfInputs);
  log('Submitted solution: ' + _currentInputValues.toString());
  return isCurrentSubmissionCorrect;
}

_createChallengeList(int numberOfElements) {
  _initialList =
      _buildDraggableItemList(_generateListOfElements(numberOfElements));
  log("Lista inicial: $_initialList");
}

_generateSteps() {
  _splitSortableItemList([_initialList]);

  while (!_isEveryInternalListOneElement(_steps.last)) {
    _splitSortableItemList(_steps.last);
```

```
  }

  log("steps: " + _steps.toString());
}

_generateMergeSteps() {
  List<List<List<SortableItem>>> splitSteps = _steps;

  while (_steps.last.length > 1) {
    List<List<SortableItem>> newStep = [];
    if (splitSteps.last.length % 2 == 0) {
      int mergeTimes = (splitSteps.last.length / 2).round();
      int index = 0;

      while (mergeTimes > 0) {
        List<SortableItem> list = List.from(splitSteps.last[index])
          ..addAll(splitSteps.last[index + 1]);
        list.sort((a, b) {
          if (int.parse(a.title) > int.parse(b.title)) {
            return 1;
          } else {
            if (int.parse(a.title) < int.parse(b.title)) {
              return -1;
            }
            return 0;
          }
        });
        newStep.add(list);
        mergeTimes--;
        index += 2;
      }
    }
    _steps.add(newStep);
  }
  log('Steps after generating Merge steps: ' + _steps.toString());
}
```

```
List<SortableItem> getInitialList() => _initialList;

List<SortableItem> getFinalList() => _finalList;

List<List<List<SortableItem>>> getSteps() => _steps;

void _splitSortableItemList(List<List<SortableItem>> sortableItemList) {
  List<List<SortableItem>> newList = [];
  for (List<SortableItem> oldList in sortableItemList) {
    int listSize = oldList.length;
    if (listSize > 1) {
      if (listSize.isEven) {
        newList.add(oldList.sublist(0, (listSize / 2).floor()));
        newList.add(oldList.sublist((listSize / 2).floor()));
      } else {
        newList.add(oldList.sublist(0, (listSize / 2).floor() + 1));
        newList.add(oldList.sublist((listSize / 2).floor() + 1));
      }
    } else if (listSize == 1) {
      newList.add(oldList);
    }
  }
  _steps.add(newList);
}

_isEveryInternalListOneElement(List<List<SortableItem>> externalList) {
  int result = 0;
  for (List<SortableItem> internalList in externalList) {
    if (internalList.length == 1) {
      result++;
    }
  }
  return result == externalList.length;
}
```

```
bool _isListOrdered(List<int> list) {
  return list.isSorted((a, b) => a.compareTo(b));
}

_generateCorrectOrderOfInputs() {
  for (List<List<SortableItem>> step in _steps) {
    for (List<SortableItem> internalList in step) {
      for (SortableItem item in internalList) {
        _correctOrderOfInputs.add(int.parse(item.title));
      }
    }
  }

  log("correct order of inputs " + _correctOrderOfInputs.toString());
}

bool isInputCorrect(int number, int position) {
  if (_correctOrderOfInputs[position] == number) {
    return true;
  }
  return false;
}

void _stepsToInputs() {
  List<List<List<InputItem>>> newInputs = [];
  int index = 0;
  for (List<List<SortableItem>> step in _steps) {
    List<List<InputItem>> stepInputs = [];
    for (List<SortableItem> internalList in step) {
      List<InputItem> internalListInputs = [];
      for (SortableItem item in internalList) {
        internalListInputs.add(InputItem(
          position: index,
          color: Colors.white,
        ));
        _currentInputValues.add('');
```

```
        index++;
      }
      stepInputs.add(internalListInputs);
    }
    newInputs.add(stepInputs);
  }
  index = 0;
  inputs = newInputs;
  notifyListeners();
}

_assignOnChangeFunctionToInputs(bool clean) {
  List<List<List<InputItem>>> newInputs = [];
  for (List<List<InputItem>> inputStep in inputs) {
    List<List<InputItem>> stepInputs = [];
    for (List<InputItem> internalInputList in inputStep) {
      List<InputItem> internalListInputs = [];
      for (InputItem item in internalInputList) {
        internalListInputs.add(InputItem(
          key: item.key,
          position: item.position,
          clean: clean,
          onChangedValue: (String value) {
            _currentInputValues[item.position] = value;
          },
          color: Colors.white,
        ));
      }
      stepInputs.add(internalListInputs);
    }
    newInputs.add(stepInputs);
  }
  inputs = newInputs;
  notifyListeners();
}
```

```
_selectWrongInputs(List<int> wrongInputPositions) {
  List<List<List<InputItem>>> newInputs = [];
  for (List<List<InputItem>> inputStep in inputs) {
    List<List<InputItem>> stepInputs = [];
    for (List<InputItem> internalInputList in inputStep) {
      List<InputItem> internalListInputs = [];
      for (InputItem item in internalInputList) {
        Color color = wrongInputPositions.contains(item.position)
            ? Colors.red
            : Colors.white;
        internalListInputs.add(InputItem(
          key: item.key,
          position: item.position,
          onChangedValue: (String value) {
            _currentInputValues[item.position] = value;
          },
          color: color,
        ));
      }
      stepInputs.add(internalListInputs);
    }
    newInputs.add(stepInputs);
  }
  inputs = newInputs;
  notifyListeners();
}

reset() {
  _steps = [];
  _correctOrderOfInputs = [];
  inputs = [];
  _currentInputValues = [];
  _createChallengeList(_numberOfElementsToSort);
  _generateSteps();
  _generateMergeSteps();
  _generateCorrectOrderOfInputs();
```

```
    _stepsToInputs();
    _assignOnChangeFunctionToInputs(true);
  }
}
```

merge_sort_3.dart

```
import 'dart:developer';
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'package:sort_it_out/src/game_modes/arcade/stages/merge_sort/
merge_sort_message_provider.dart';
import 'package:sort_it_out/src/game_modes/arcade/stages/merge_sort/stage_3/
merge_sort_provider_3.dart';
import 'package:sort_it_out/src/model/sortable_item.dart';
import 'package:sort_it_out/src/save_data/save_data_provider.dart';
import 'package:sort_it_out/src/score_system/score_system_provider.dart';

class MergeSort3 extends StatefulWidget {
  const MergeSort3({Key? key}) : super(key: key);

  @override
  State<StatefulWidget> createState() => _MergeSort3State();
}

class _MergeSort3State extends State<MergeSort3> {
  @override
  Widget build(BuildContext context) {
    MergeSortProvider3 _mergeSortProvider =
        Provider.of<MergeSortProvider3>(context);
    ScoreSystemProvider _scoreSystemProvider =
        Provider.of<ScoreSystemProvider>(context);
    MergeSortMessageProvider _messageProvider =
        Provider.of<MergeSortMessageProvider>(context);
    SaveDataProvider _saveProvider = Provider.of<SaveDataProvider>(context);
    return Scaffold(
```

```
appBar: AppBar(
  title: const Text('Algoritmo 1 - Merge Sort'),
  centerTitle: true,
),
body: Padding(
  padding: const EdgeInsets.only(left: 8.0),
  child: Row(
    mainAxisAlignment: MainAxisAlignment.spaceBetween,
    children: [
      Column(
        crossAxisAlignment: CrossAxisAlignment.center,
        mainAxisAlignment: MainAxisAlignment.start,
        children: [
          Column(
            children: [
              const SizedBox(height: 20),
              Padding(
                padding: const EdgeInsets.all(8.0),
                child: Text(
                    'Score: ' + _scoreSystemProvider.score.toString()),
              )
            ],
          ),
          Padding(
            padding: const EdgeInsets.all(16.0),
            child: TextButton(
                onPressed: () {
                  {
                    Future.delayed(const Duration(seconds: 1));
                    _hintAsked(_scoreSystemProvider, _messageProvider);
                  }
                },
                child: const Text('Pedir dica (-20 pontos)')),
          ),
          SizedBox(
            width: 220,
```

```
      child: Text(
        _messageProvider.currentMessage,
        softWrap: true,
      ),
    ),
  ],
),
SingleChildScrollView(
  child: SizedBox(
    child: Column(
      mainAxisAlignment: MainAxisAlignment.center,
      children: [
        ConstrainedBox(
          constraints: const BoxConstraints(
            maxHeight: 60,
            minWidth: 150,
          ),
          child: ListView(
              padding: const EdgeInsets.only(bottom: 10),
              shrinkWrap: true,
              scrollDirection: Axis.horizontal,
              children: _mergeSortProvider.getInitialList()),
        ),
        for (List<List<SortableItem>> step
            in _mergeSortProvider.getSteps())
          Column(
            children: [
              Padding(
                padding: const EdgeInsets.only(bottom: 20),
                child: Column(
                  children: [
                    Row(
                      children: [
                        for (List<SortableItem> internalList
                            in step)
                          step.last != internalList
```

```
            ? Row(
                children: [
                  IconButton(
                      onPressed: () =>
                          _selectInternalList(
                              internalList,
                              _mergeSortProvider,
                              _messageProvider,
                              _saveProvider,
                              _scoreSystemProvider),
                      icon: const Icon(Icons
                          .arrow_downward_sharp)),
                  SizedBox(
                    width: step.length == 4
                        ? 40
                        : 40 / step.length,
                  )
                ],
              )
            : IconButton(
                onPressed: () =>
                    _selectInternalList(
                        internalList,
                        _mergeSortProvider,
                        _messageProvider,
                        _saveProvider,
                        _scoreSystemProvider),
                icon: const Icon(
                    Icons.arrow_downward_sharp)),
        const SizedBox(
          width: 36,
        )
      ],
    ),
    Row(
      mainAxisAlignment: MainAxisAlignment.center,
```

```
                        children: _buildWholeStep(step),
                      ),
                    ],
                  ),
                ),
              ),
            ],
          ),
        ),
      ),
      const SizedBox(
        width: 16,
      )
    ],
  ),
 ),
);
}

_buildWholeStep(List<List<SortableItem>> step) {
 List<Widget> list = [];
 for (int i = 0; i < step.length; i++) {
  list.add(Padding(
   padding: i == step.length - 1
     ? const EdgeInsets.only(right: 20)
     : EdgeInsets.zero,
   child: Row(
    children: [
     SizedBox(
      width: step[i].length * 40,
      height: 40,
      child: ListView(
       scrollDirection: Axis.horizontal,
       children: step[i],
      ),
```

```
          ),
          const SizedBox(
            width: 10,
          )
        ],
      ),
    ));
  }
  return list;
}

_hintAsked(ScoreSystemProvider scoreSystemProvider,
    MergeSortMessageProvider messageProvider) {
  scoreSystemProvider.hintAsked();
  messageProvider.hintAsked();
}

_selectInternalList(
    List<SortableItem> internalList,
    MergeSortProvider3 mergeSortProvider,
    MergeSortMessageProvider messageProvider,
    SaveDataProvider saveDataProvider,
    ScoreSystemProvider scoreProvider) {
  bool result = mergeSortProvider.checkInternalListSelection(internalList);

  result
      ? _correctSelection(messageProvider, mergeSortProvider,
          saveDataProvider, scoreProvider, internalList)
      : _incorrectSelection(
          messageProvider, mergeSortProvider, scoreProvider);
  log('selected internalList: ${internalList.toString()} is it correct? [$result]');
}

_correctSelection(
    MergeSortMessageProvider messageProvider,
    MergeSortProvider3 mergeSortProvider,
```

```
    SaveDataProvider saveDataProvider,
    ScoreSystemProvider scoreProvider,
    List<SortableItem> internalList) {
  messageProvider.correctMoveMessage();
  scoreProvider.correctMove();
  mergeSortProvider.select(internalList);
  if (mergeSortProvider.isStageSolved(internalList)) {
    _onStageSolved(mergeSortProvider, saveDataProvider);
  }
}

_incorrectSelection(MergeSortMessageProvider messageProvider,
    MergeSortProvider3 mergeSortProvider, ScoreSystemProvider scoreProvider) {
  scoreProvider.wrongMove();
  messageProvider.wrongMoveMessage();
}

_onStageSolved(
    MergeSortProvider3 mergeSortProvider, SaveDataProvider saveProvider) {
  showDialog(
      context: context,
      builder: (BuildContext context) => AlertDialog(
          title: const Text('Sucesso!'),
          content: const Text('Parabéns! Você resolveu o desafio!'),
          actions: <Widget>[
            TextButton(
              onPressed: () {
                saveProvider.saveData!.mergeSortSaveData.isStage3Complete =
                    true;
                saveProvider.saveStageCompletion();
                mergeSortProvider.reset();
                Navigator.pop(context, 'OK');
              },
              child: const Text('OK'),
            ),
          ],
```

```
        ));
  }
}
```

merge_sort_provider_3.dart

```
import 'dart:developer';

import 'package:flutter/foundation.dart';
import 'package:flutter/material.dart';
import 'package:sort_it_out/src/model/sortable_item.dart';
import 'package:collection/collection.dart';

class MergeSortProvider3 extends ChangeNotifier {
  late List<SortableItem> _initialList;
  late List<List<List<SortableItem>>> steps;
  late List<SortableItem> _finalList;
  final int _numberOfElementsToSort = 7;
  late List<List<SortableItem>> _selectedInternalLists;
  late List<List<SortableItem>> _correctOrderOfInternalListsSelection;

  MergeSortProvider3() {
    steps = [];
    _selectedInternalLists = [];
    _correctOrderOfInternalListsSelection = [];
    _createChallengeList(_numberOfElementsToSort);
    _generateSteps();
    _generateMergeSteps();
    _generateCorrectOrderOfInternalListsSelection();
  }

  List<int> _generateListOfElements(int listSize) {
    List<int> listOfRandomNumbers =
        List.generate(listSize * 2, (index) => index);
    listOfRandomNumbers.shuffle();
    List<int> returnList = listOfRandomNumbers.sublist(0, listSize);
```

```
  while (_isListOrdered(returnList)) {
    listOfRandomNumbers.shuffle();
    returnList = listOfRandomNumbers.sublist(0, listSize);
  }

  return returnList;
}

List<SortableItem> _buildDraggableItemList(List<int> randomNumbers) {
  List<SortableItem> draggableItemList = [];
  for (int number in randomNumbers) {
    SortableItem item = SortableItem(
      key: ValueKey(number),
      color: Colors.white,
      title: number.toString(),
      height: 20.0,
      width: 30.0,
      fontSize: 15,
    );
    draggableItemList.add(item);
  }
  return draggableItemList;
}

bool checkInternalListSelection(List<SortableItem> internalList) {
  bool isCorrect = isCorrectlySelected(internalList);
  if (isCorrect) {
    _selectedInternalLists.add(internalList);
  }
  return isCorrect;
}

_createChallengeList(int numberOfElements) {
  _initialList =
      _buildDraggableItemList(_generateListOfElements(numberOfElements));
```

```
  log("Lista inicial: $_initialList");
}

_generateSteps() {
  _splitSortableItemList([_initialList]);

  while (!_isEveryInternalListOneElement(steps.last)) {
    _splitSortableItemList(steps.last);
  }

  log("steps: " + steps.toString());
}

_generateMergeSteps() {
  List<List<List<SortableItem>>> splitSteps = steps;

  while (steps.last.length > 1) {
    List<List<SortableItem>> newStep = [];
    int mergeTimes = (splitSteps.last.length / 2).floor();
    int index = 0;

    while (mergeTimes > 0) {
      // if (index + 1 <= splitSteps.last.length) {
      List<SortableItem> list = List.from(splitSteps.last[index])
        ..addAll(splitSteps.last[index + 1]);
      list.sort((a, b) {
        if (int.parse(a.title) > int.parse(b.title)) {
          return 1;
        } else {
          if (int.parse(a.title) < int.parse(b.title)) {
            return -1;
          }
          return 0;
        }
      });
      newStep.add(list);
```

```
      mergeTimes--;
      if (index + 1 != _numberOfElementsToSort - 2) {
        index += 2;
      } else {
        newStep.add(splitSteps.last[index + 2]);
      }
      // } else {
      //   newStep.add(splitSteps.last[index]);
      // }
    }

    steps.add(newStep);
  }
  log('Steps after generating Merge steps: ' + steps.toString());
}

List<SortableItem> getInitialList() => _initialList;

List<SortableItem> getFinalList() => _finalList;

List<List<List<SortableItem>>> getSteps() => steps;

void _splitSortableItemList(List<List<SortableItem>> sortableItemList) {
  List<List<SortableItem>> newList = [];
  for (List<SortableItem> oldList in sortableItemList) {
    int listSize = oldList.length;
    if (listSize > 1) {
      if (listSize.isEven) {
        newList.add(oldList.sublist(0, (listSize / 2).floor()));
        newList.add(oldList.sublist((listSize / 2).floor()));
      } else {
        newList.add(oldList.sublist(0, (listSize / 2).floor() + 1));
        newList.add(oldList.sublist((listSize / 2).floor() + 1));
      }
    } else if (listSize == 1) {
      newList.add(oldList);
```

```
    }
  }
  steps.add(newList);
}

_isEveryInternalListOneElement(List<List<SortableItem>> externalList) {
  int result = 0;
  for (List<SortableItem> internalList in externalList) {
    if (internalList.length == 1) {
      result++;
    }
  }
  return result == externalList.length;
}

bool _isListOrdered(List<int> list) {
  return list.isSorted((a, b) => a.compareTo(b));
}

void _generateCorrectOrderOfInternalListsSelection() {
  _correctOrderOfInternalListsSelection.addAll([
    steps[0][0],
    steps[1][0],
    steps[2][0],
    steps[2][1],
    steps[3][0],
    steps[1][1],
    steps[2][2],
    steps[2][3],
    steps[3][1],
    steps[4][0],
    steps[0][1],
    steps[1][2],
    steps[2][4],
    steps[2][5],
    steps[3][2],
```

```
    steps[1][3],
    steps[2][6],
    steps[3][3],
    steps[4][1],
    steps[5][0]
  ]);

  log('correct order of internal lists: ' +
      _correctOrderOfInternalListsSelection.toString());
}

bool isCorrectlySelected(List<SortableItem> internalList) {
  if (_correctOrderOfInternalListsSelection.contains(internalList)) {
    if (_correctOrderOfInternalListsSelection[
          _selectedInternalLists.length] ==
        internalList) {
      return true;
    }
  }
  return false;
}

reset() {
  steps = [];
  _selectedInternalLists = [];
  _correctOrderOfInternalListsSelection = [];
  _createChallengeList(_numberOfElementsToSort);
  _generateSteps();
  _generateMergeSteps();
  _generateCorrectOrderOfInternalListsSelection();
}

bool isStageSolved(List<SortableItem> currentInternalList) {
  if (currentInternalList.length != steps.last.last.length) {
    return false;
  }
```

```
  int index = 0;
  for (SortableItem item in steps.last.last) {
    if (item.title != currentInternalList[index].title) {
      return false;
    }
    index++;
  }
  return true;
}

int _getLevelOfInternalList(List<SortableItem> selectedInternalList) {
  int outerIndex = 0;
  for (List<List<SortableItem>> step in steps) {
    for (List<SortableItem> internalList in step) {
      if (listEquals(internalList, selectedInternalList)) {
        return outerIndex;
      }
    }
    outerIndex++;
  }

  return -1;
}

void select(List<SortableItem> selectedInternalList) {
  int level = _getLevelOfInternalList(selectedInternalList);

  int index = 0;
  List<SortableItem> newInternalList = [];
  for (List<SortableItem> internalList in steps[level]) {
    if (listEquals(internalList, selectedInternalList)) {
      List<SortableItem> newInternalList = [];
      for (SortableItem item in steps[level][index]) {
        newInternalList.add(SortableItem(
            title: item.title,
            color: Colors.green,
```

```
            key: item.key,
            height: 20.0,
            width: 30.0,
            fontSize: 15));
       }

       steps[level][index] = newInternalList;
     }

     index++;
   }

   notifyListeners();
 }
}
```

merge_sort_4.dart

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'package:sort_it_out/src/game_modes/arcade/stages/merge_sort/
merge_sort_message_provider.dart';
import 'package:sort_it_out/src/game_modes/arcade/stages/merge_sort/stage_4/
merge_sort_provider_4.dart';
import 'package:sort_it_out/src/model/input_item.dart';
import 'package:sort_it_out/src/save_data/save_data_provider.dart';
import 'package:sort_it_out/src/score_system/score_system_provider.dart';

class MergeSort4 extends StatefulWidget {
  const MergeSort4({Key? key}) : super(key: key);

  @override
  State<StatefulWidget> createState() => _MergeSort4State();
}

class _MergeSort4State extends State<MergeSort4> {
```

```
@override
Widget build(BuildContext context) {
  MergeSortProvider4 _mergeSortProvider =
      Provider.of<MergeSortProvider4>(context);
  ScoreSystemProvider _scoreSystemProvider =
      Provider.of<ScoreSystemProvider>(context);
  MergeSortMessageProvider _messageProvider =
      Provider.of<MergeSortMessageProvider>(context);
  SaveDataProvider _saveProvider = Provider.of<SaveDataProvider>(context);
  return Scaffold(
    appBar: AppBar(
      title: const Text('Algoritmo 2 - Merge Sort'),
      centerTitle: true,
    ),
    body: Padding(
      padding: const EdgeInsets.only(left: 8.0),
      child: Row(
        mainAxisAlignment: MainAxisAlignment.spaceBetween,
        children: [
          SingleChildScrollView(
            child: Column(
              crossAxisAlignment: CrossAxisAlignment.center,
              mainAxisAlignment: MainAxisAlignment.start,
              children: [
                Column(
                  children: [
                    const SizedBox(height: 20),
                    Padding(
                      padding: const EdgeInsets.all(8.0),
                      child: Text(
                          'Score: ' + _scoreSystemProvider.score.toString()),
                    )
                  ],
                ),
                Padding(
                  padding: const EdgeInsets.all(16.0),
```

```
        child: TextButton(
            onPressed: () {
              {
                Future.delayed(const Duration(seconds: 1));
                _hintAsked(_scoreSystemProvider, _messageProvider);
              }
            },
            child: const Text('Pedir dica (-20 pontos)')),
      ),
      SizedBox(
        width: 270,
        child: Text(
          _messageProvider.currentMessage,
          softWrap: true,
        ),
      ),
      Padding(
        padding: const EdgeInsets.all(16.0),
        child: TextButton(
            onPressed: () {
              {
                Future.delayed(const Duration(seconds: 1));
                _submitInputValues(
                    _mergeSortProvider,
                    _saveProvider,
                    _scoreSystemProvider,
                    _messageProvider);
              }
            },
            child: const Text('Submeter')),
      ),
    ],
  ),
),
SingleChildScrollView(
  child: SizedBox(
```

```
child: Column(
  mainAxisAlignment: MainAxisAlignment.center,
  children: [
    ConstrainedBox(
      constraints: const BoxConstraints(
        maxHeight: 60,
        minWidth: 150,
      ),
      child: ListView(
          padding: const EdgeInsets.only(bottom: 10),
          shrinkWrap: true,
          scrollDirection: Axis.horizontal,
          children: _mergeSortProvider.getInitialList()),
    ),
    for (List<List<InputItem>> step
        in _mergeSortProvider.inputs)
      Column(
        children: [
          Padding(
            padding: const EdgeInsets.only(bottom: 20),
            child: Column(
              children: [
                Row(children: [
                  const SizedBox(
                    width: 20,
                  ),
                  for (List<InputItem> internalList in step)
                    step.last != internalList
                        ? Row(
                            children: [
                              const Icon(
                                  Icons.arrow_downward_sharp),
                              SizedBox(
                                width: 200 / step.length,
                              )
                            ],
```

```
                                )
                                : const Icon(
                                    Icons.arrow_downward_sharp),
                            const SizedBox(
                              width: 30,
                            )
                          ]),
                          Row(
                            mainAxisAlignment: MainAxisAlignment.center,
                            children: _buildWholeStep(step),
                          ),
                        ],
                      ),
                    ),
                  ],
                ),
              ],
            ),
          ),
          const SizedBox(
            width: 16,
          )
        ],
      ),
    ),
  );
}

_buildWholeStep(List<List<InputItem>> step) {
  List<Widget> list = [];

  for (int i = 0; i < step.length; i++) {
    list.add(Padding(
      padding: i == step.length - 1
          ? const EdgeInsets.only(right: 20)
```

```
          : EdgeInsets.zero,
      child: Row(
        children: [
          SizedBox(
            width: step[i].length * 40,
            height: 40,
            child: ListView(
              scrollDirection: Axis.horizontal,
              children: step[i],
            ),
          ),
          const SizedBox(
            width: 10,
          )
        ],
      ),
    ));
  }
  return list;
}

_hintAsked(ScoreSystemProvider scoreSystemProvider,
    MergeSortMessageProvider messageProvider) {
  scoreSystemProvider.hintAsked();
  messageProvider.hintAsked();
}

_submitInputValues(
  MergeSortProvider4 mergeSortProvider,
  SaveDataProvider saveDataProvider,
  ScoreSystemProvider scoreProvider,
  MergeSortMessageProvider messageProvider,
) {
  bool isSubmissionCorrect = mergeSortProvider.checkInputs();
  if (isSubmissionCorrect) {
    _correctInputs(
```

```
        messageProvider, mergeSortProvider, saveDataProvider, scoreProvider);
    } else {
      if (_areThereEmptyInputs(mergeSortProvider)) {
        _emptyInputs(messageProvider, mergeSortProvider);
      } else {
        _incorrectInputs(messageProvider, mergeSortProvider, scoreProvider);
      }
    }
  }

  _areThereEmptyInputs(MergeSortProvider4 mergeSortProvider) {
    return mergeSortProvider.areThereEmptyInputs();
  }

  _correctInputs(
      MergeSortMessageProvider messageProvider,
      MergeSortProvider4 mergeSortProvider,
      SaveDataProvider saveDataProvider,
      ScoreSystemProvider scoreProvider) {
    messageProvider.correctMoveMessage();
    scoreProvider.correctMove();
    _onStageSolved(mergeSortProvider, saveDataProvider);
  }

  _incorrectInputs(MergeSortMessageProvider messageProvider,
      MergeSortProvider4 mergeSortProvider, ScoreSystemProvider scoreProvider) {
    scoreProvider.wrongMove();
    messageProvider.wrongMoveMessageInputs();
    mergeSortProvider.highlightWrongInputs();
  }

  _emptyInputs(MergeSortMessageProvider messageProvider,
      MergeSortProvider4 mergeSortProvider) {
    // --- perguntar se deve fazer o usuario perder pontos
    messageProvider.emptyInputsErrorMessage();
  }
```

```
  _onStageSolved(
      MergeSortProvider4 mergeSortProvider, SaveDataProvider saveProvider) {
    showDialog(
        context: context,
        builder: (BuildContext context) => AlertDialog(
            title: const Text('Sucesso!'),
            content: const Text('Parabéns! Você resolveu o desafio!'),
            actions: <Widget>[
              TextButton(
                onPressed: () {
                  saveProvider.saveData!.mergeSortSaveData.isStage2Complete =
                      true;
                  saveProvider.saveStageCompletion();
                  mergeSortProvider.reset();
                  Navigator.pop(context, 'OK');
                },
                child: const Text('OK'),
              ),
            ],
          ));
  }
}
```

merge_sort_provider_4.dart

```
import 'dart:developer';
import 'package:flutter/foundation.dart';
import 'package:flutter/material.dart';
import 'package:sort_it_out/src/model/input_item.dart';
import 'package:sort_it_out/src/model/sortable_item.dart';
import 'package:collection/collection.dart';

class MergeSortProvider4 extends ChangeNotifier {
  late List<SortableItem> _initialList;
  late List<List<List<SortableItem>>> _steps;
```

```
late List<SortableItem> _finalList;
bool isStageSolved = false;
final int _numberOfElementsToSort = 7;
late List<int> _correctOrderOfInputs;
late List<List<List<InputItem>>> inputs;
late List<String> _currentInputValues;

MergeSortProvider4() {
  _steps = [];
  _correctOrderOfInputs = [];
  inputs = [];
  _currentInputValues = [];
  _createChallengeList(_numberOfElementsToSort);
  _generateSteps();
  _generateMergeSteps();
  _generateCorrectOrderOfInputs();
  _stepsToInputs();
  _assignOnChangeFunctionToInputs(true);
}

List<int> _generateListOfElements(int listSize) {
  List<int> listOfRandomNumbers =
      List.generate(listSize * 2, (index) => index);
  listOfRandomNumbers.shuffle();
  List<int> returnList = listOfRandomNumbers.sublist(0, listSize);

  while (_isListOrdered(returnList)) {
    listOfRandomNumbers.shuffle();
    returnList = listOfRandomNumbers.sublist(0, listSize);
  }

  return returnList;
}

List<SortableItem> _buildDraggableItemList(List<int> randomNumbers) {
  List<SortableItem> draggableItemList = [];
```

```
  for (int number in randomNumbers) {
    SortableItem item = SortableItem(
      key: ValueKey(number),
      color: Colors.white,
      title: number.toString(),
      height: 20.0,
      width: 30.0,
      fontSize: 15,
    );
    draggableItemList.add(item);
  }
  return draggableItemList;
}

highlightWrongInputs() {
  int index = 0;
  List<int> wrongInputPositions = [];
  for (String element in _currentInputValues) {
    if (int.parse(element) != _correctOrderOfInputs[index]) {
      wrongInputPositions.add(index);
    }
    index++;
  }

  log('Correct inputs in order: ${_correctOrderOfInputs.toString()}');
  log('These elements are wrong: ${wrongInputPositions.toString()}');

  _selectWrongInputs(wrongInputPositions);
}

bool areThereEmptyInputs() {
  List<int> emptyInputsPositions = [];
  int index = 0;
  for (String inputValue in _currentInputValues) {
    if (inputValue.isEmpty) {
      emptyInputsPositions.add(index);
```

```
    }
    index++;
  }

  if (emptyInputsPositions.isEmpty) {
    return false;
  } else {
    _selectWrongInputs(emptyInputsPositions);
    return true;
  }
}

bool checkInputs() {
  List<int> currentValuesToIntList = [];
  for (String inputValue in _currentInputValues) {
    currentValuesToIntList.add(int.tryParse(inputValue) ?? -1);
  }
  bool isCurrentSubmissionCorrect =
      listEquals(currentValuesToIntList, _correctOrderOfInputs);
  log('Submitted solution: ' + _currentInputValues.toString());
  return isCurrentSubmissionCorrect;
}

_createChallengeList(int numberOfElements) {
  _initialList =
      _buildDraggableItemList(_generateListOfElements(numberOfElements));
  log("Lista inicial: $_initialList");
}

_generateSteps() {
  _splitSortableItemList([_initialList]);

  while (!_isEveryInternalListOneElement(_steps.last)) {
    _splitSortableItemList(_steps.last);
  }
```

```
  log("steps: " + _steps.toString());
}

_generateMergeSteps() {
  List<List<List<SortableItem>>> splitSteps = _steps;

  while (_steps.last.length > 1) {
    List<List<SortableItem>> newStep = [];
    int mergeTimes = (splitSteps.last.length / 2).floor();
    int index = 0;

    while (mergeTimes > 0) {
      List<SortableItem> list = List.from(splitSteps.last[index])
        ..addAll(splitSteps.last[index + 1]);
      list.sort((a, b) {
        if (int.parse(a.title) > int.parse(b.title)) {
          return 1;
        } else {
          if (int.parse(a.title) < int.parse(b.title)) {
            return -1;
          }
          return 0;
        }
      });
      newStep.add(list);
      mergeTimes--;
      if (index + 1 != _numberOfElementsToSort - 2) {
        index += 2;
      } else {
        newStep.add(splitSteps.last[index + 2]);
      }
    }

    _steps.add(newStep);
  }
  log('Steps after generating Merge steps: ' + _steps.toString());
```

```
}

List<SortableItem> getInitialList() => _initialList;

List<SortableItem> getFinalList() => _finalList;

List<List<List<SortableItem>>> getSteps() => _steps;

void _splitSortableItemList(List<List<SortableItem>> sortableItemList) {
  List<List<SortableItem>> newList = [];
  for (List<SortableItem> oldList in sortableItemList) {
    int listSize = oldList.length;
    if (listSize > 1) {
      if (listSize.isEven) {
        newList.add(oldList.sublist(0, (listSize / 2).floor()));
        newList.add(oldList.sublist((listSize / 2).floor()));
      } else {
        newList.add(oldList.sublist(0, (listSize / 2).floor() + 1));
        newList.add(oldList.sublist((listSize / 2).floor() + 1));
      }
    } else if (listSize == 1) {
      newList.add(oldList);
    }
  }
  _steps.add(newList);
}

_isEveryInternalListOneElement(List<List<SortableItem>> externalList) {
  int result = 0;
  for (List<SortableItem> internalList in externalList) {
    if (internalList.length == 1) {
      result++;
    }
  }
  return result == externalList.length;
}
```

```
bool _isListOrdered(List<int> list) {
  return list.isSorted((a, b) => a.compareTo(b));
}

_generateCorrectOrderOfInputs() {
  for (List<List<SortableItem>> step in _steps) {
    for (List<SortableItem> internalList in step) {
      for (SortableItem item in internalList) {
        _correctOrderOfInputs.add(int.parse(item.title));
      }
    }
  }

  log("correct order of inputs " + _correctOrderOfInputs.toString());
}

bool isInputCorrect(int number, int position) {
  if (_correctOrderOfInputs[position] == number) {
    return true;
  }
  return false;
}

void _stepsToInputs() {
  List<List<List<InputItem>>> newInputs = [];
  int index = 0;
  for (List<List<SortableItem>> step in _steps) {
    List<List<InputItem>> stepInputs = [];
    for (List<SortableItem> internalList in step) {
      List<InputItem> internalListInputs = [];
      for (SortableItem item in internalList) {
        internalListInputs.add(InputItem(
          position: index,
          color: Colors.white,
        ));
```

```
        _currentInputValues.add('');
        index++;
      }
      stepInputs.add(internalListInputs);
    }
    newInputs.add(stepInputs);
  }
  index = 0;
  inputs = newInputs;
  notifyListeners();
}

_assignOnChangeFunctionToInputs(bool clean) {
  List<List<List<InputItem>>> newInputs = [];
  for (List<List<InputItem>> inputStep in inputs) {
    List<List<InputItem>> stepInputs = [];
    for (List<InputItem> internalInputList in inputStep) {
      List<InputItem> internalListInputs = [];
      for (InputItem item in internalInputList) {
        internalListInputs.add(InputItem(
          key: item.key,
          position: item.position,
          clean: clean,
          onChangedValue: (String value) {
            _currentInputValues[item.position] = value;
          },
          color: Colors.white,
        ));
      }
      stepInputs.add(internalListInputs);
    }
    newInputs.add(stepInputs);
  }
  inputs = newInputs;
  notifyListeners();
}
```

```
_selectWrongInputs(List<int> wrongInputPositions) {
  List<List<List<InputItem>>> newInputs = [];
  for (List<List<InputItem>> inputStep in inputs) {
    List<List<InputItem>> stepInputs = [];
    for (List<InputItem> internalInputList in inputStep) {
      List<InputItem> internalListInputs = [];
      for (InputItem item in internalInputList) {
        Color color = wrongInputPositions.contains(item.position)
            ? Colors.red
            : Colors.white;
        internalListInputs.add(InputItem(
          key: item.key,
          position: item.position,
          onChangedValue: (String value) {
            _currentInputValues[item.position] = value;
          },
          color: color,
        ));
      }
      stepInputs.add(internalListInputs);
    }
    newInputs.add(stepInputs);
  }
  inputs = newInputs;
  notifyListeners();
}

reset() {
  _steps = [];
  _correctOrderOfInputs = [];
  inputs = [];
  _currentInputValues = [];
  _createChallengeList(_numberOfElementsToSort);
  _generateSteps();
  _generateMergeSteps();
```

```
    _generateCorrectOrderOfInputs();
    _stepsToInputs();
    _assignOnChangeFunctionToInputs(true);
  }
}
```

merge_sort_message_provider.dart

```
import 'package:flutter/cupertino.dart';

class MergeSortMessageProvider extends ChangeNotifier {
  late String currentMessage;

  late List<String> _hints;

  MergeSortMessageProvider() {
    currentMessage = '';
    _hints = [
      'Lembre-se que não há duas fases distintas nesse algoritmo',
      'A lista inicial é dividida em duas partes, tente organizar uma das partes antes de
seguir para a outra'
    ];
  }

  void wrongMoveMessage() {
    currentMessage = 'Seleção incorreta!';
    notifyListeners();
  }

  void wrongMoveMessageInputs() {
    currentMessage = 'Alguns elementos estão errados!';
    notifyListeners();
  }

  void emptyInputsErrorMessage() {
    currentMessage = 'Alguns campos estão vazios!';
```

```
    notifyListeners();
  }

  void correctMoveMessage() {
    currentMessage = 'Bom trabalho!';
    notifyListeners();
  }

  void hintAsked() {
    currentMessage = getRandomHint();
    notifyListeners();
  }

  String getRandomHint() {
    List<int> listOfRandomNumbers =
        List.generate(_hints.length, (index) => index);
    listOfRandomNumbers.shuffle();
    return _hints[listOfRandomNumbers.first];
  }
}
```

arcade_menu.dart

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'package:sort_it_out/src/game_modes/arcade/stages/bubble_sort/stage_1/
bubble_sort_1.dart';
import 'package:sort_it_out/src/game_modes/arcade/stages/bubble_sort/stage_2/
bubble_sort_2.dart';
import 'package:sort_it_out/src/game_modes/arcade/stages/bubble_sort/stage_3/
bubble_sort_3.dart';
import 'package:sort_it_out/src/game_modes/arcade/stages/merge_sort/stage_1/
merge_sort_1.dart';
import 'package:sort_it_out/src/game_modes/arcade/stages/merge_sort/stage_2/
merge_sort_2.dart';
```

```
import 'package:sort_it_out/src/game_modes/arcade/stages/merge_sort/stage_3/
merge_sort_3.dart';
import 'package:sort_it_out/src/game_modes/arcade/stages/merge_sort/stage_4/
merge_sort_4.dart';
import 'package:sort_it_out/src/save_data/save_data_provider.dart';

class ArcadeMenu extends StatefulWidget {
  const ArcadeMenu({Key? key, required this.title}) : super(key: key);

  final String title;

  @override
  State<ArcadeMenu> createState() => _ArcadeMenuState();
}

class _ArcadeMenuState extends State<ArcadeMenu> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(widget.title),
      ),
      body: SingleChildScrollView(
        child: Consumer<SaveDataProvider>(
          builder: ((context, saveProvider, child) => Center(
                child: Column(
                  mainAxisAlignment: MainAxisAlignment.center,
                  children: [
                    const SizedBox(
                      height: 30,
                    ),
                    const Text('Bubble Sort'),
                    TextButton(
                        child: const Text('Fase 1'),
                        onPressed: () {
                          Navigator.push(
```

```
        context,
        MaterialPageRoute(
            builder: (context) => const BubbleSort1()),
      );
    },
    style: Theme.of(context).textButtonTheme.style),
if (saveProvider.bubbleSortSaveData.isStage1Complete)
  TextButton(
      child: const Text('Fase 2'),
      onPressed: () {
        Navigator.push(
          context,
          MaterialPageRoute(
              builder: (context) => const BubbleSort2()),
        );
      },
      style: Theme.of(context).textButtonTheme.style),
if (saveProvider.bubbleSortSaveData.isStage2Complete)
  TextButton(
      child: const Text('Fase 3'),
      onPressed: () {
        Navigator.push(
          context,
          MaterialPageRoute(
              builder: (context) => const BubbleSort3()),
        );
      },
      style: Theme.of(context).textButtonTheme.style),
if (saveProvider.bubbleSortSaveData.isStage3Complete) ...[
  const Text('Merge Sort'),
  TextButton(
      child: const Text('Fase 1'),
      onPressed: () {
        Navigator.push(
          context,
          MaterialPageRoute(
```

```
            builder: (context) => const MergeSort1()),
      );
    },
    style: Theme.of(context).textButtonTheme.style),
],
if (saveProvider.mergeSortSaveData.isStage1Complete)
  TextButton(
    child: const Text('Fase 2'),
    onPressed: () {
      Navigator.push(
        context,
        MaterialPageRoute(
            builder: (context) => const MergeSort2()),
      );
    },
    style: Theme.of(context).textButtonTheme.style),
if (saveProvider.mergeSortSaveData.isStage2Complete)
  TextButton(
    child: const Text('Fase 3'),
    onPressed: () {
      Navigator.push(
        context,
        MaterialPageRoute(
            builder: (context) => const MergeSort3()),
      );
    },
    style: Theme.of(context).textButtonTheme.style),
if (saveProvider.mergeSortSaveData.isStage3Complete)
  TextButton(
    child: const Text('Fase 4'),
    onPressed: () {
      Navigator.push(
        context,
        MaterialPageRoute(
            builder: (context) => const MergeSort4()),
      );
```

```
                    },
                    style: Theme.of(context).textButtonTheme.style),
              ],
            ),
          )),
      ),
    ),
  );
 }
}
```

input_item.dart

```
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';

class InputItem extends StatefulWidget {
  final int position;
  final Color color;
  final double height;
  final double width;
  final double fontSize;
  final onChangedValue;
  final bool clean;

  const InputItem(
      {Key? key,
      required this.position,
      required this.color,
      this.onChangedValue,
      this.clean = false,
      this.height = 40.0,
      this.width = 40.0,
      this.fontSize = 20.0})
      : super(key: key);
```

```
  @override
  State<StatefulWidget> createState() => InputItemState();
}

class InputItemState extends State<InputItem> {
  String value = '';

  @override
  Widget build(BuildContext context) {
    return ClipRRect(
      borderRadius: BorderRadius.circular(10.0),
      child: Container(
        color: widget.color,
        width: widget.width,
        height: widget.height,
        child: Center(
            child: TextField(
                textAlign: TextAlign.center,
                onChanged: (value) => widget.onChangedValue(value),
                controller:
                    widget.clean ? TextEditingController(text: "") : null,
                keyboardType: TextInputType.number,
                inputFormatters: <TextInputFormatter>[
              FilteringTextInputFormatter.digitsOnly
            ])),
      ),
    );
  }
}
```

sortable_item.dart

```
import 'package:flutter/material.dart';

class SortableItem extends StatefulWidget {
  final String title;
```

```
  final Color color;
  final double height;
  final double width;
  final double fontSize;

  const SortableItem(
      {Key? key,
      required this.title,
      required this.color,
      this.height = 40.0,
      this.width = 40.0,
      this.fontSize = 20.0})
      : super(key: key);

  @override
  State<StatefulWidget> createState() => SortableItemState();
}

class SortableItemState extends State<SortableItem> {
  @override
  Widget build(BuildContext context) {
    return ClipRRect(
      borderRadius: BorderRadius.circular(10.0),
      child: Container(
        color: widget.color,
        width: widget.width,
        height: widget.height,
        child: Center(
          child: Text(
            widget.title,
            style: TextStyle(fontSize: widget.fontSize),
          ),
        ),
      ),
    );
  }
```

```
}
```

options_menu.dart

```
import 'package:flutter/material.dart';
import 'package:provider/provider.dart';
import 'package:sort_it_out/src/save_data/save_data_provider.dart';

class OptionsMenu extends StatefulWidget {
  const OptionsMenu({Key? key, required this.title}) : super(key: key);

  final String title;

  @override
  State<OptionsMenu> createState() => _OptionsMenuState();
}

class _OptionsMenuState extends State<OptionsMenu> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(widget.title),
      ),
      body: SingleChildScrollView(
        child: Consumer<SaveDataProvider>(
          builder: ((context, saveProvider, child) => Center(
                child: Column(
                  mainAxisAlignment: MainAxisAlignment.center,
                  children: [
                    const SizedBox(
                      height: 20,
                    ),
                    TextButton(
                        child: const Text('Apagar dados de progresso salvos'),
                        onPressed: () {
```

```
      showDialog(
          context: context,
          builder: (BuildContext context) => AlertDialog(
              title: const Text('Cuidado!'),
              content: const Text(
                  'Você realmente deseja resetar seus dados salvos?'),
              actions: <Widget>[
                TextButton(
                  onPressed: () {
                    saveProvider.resetSaveData();
                    Navigator.pop(context, 'OK');
                  },
                  child: const Text('Sim'),
                ),
                TextButton(
                  onPressed: () {
                    Navigator.pop(context, 'Canceled');
                  },
                  child: const Text('Cancelar'),
                ),
              ],
            ));
    },
    style: Theme.of(context).textButtonTheme.style),
if (saveProvider.saveData != null)
  TextButton(
      child: const Text(
          'Localizar seu arquivo de dados de progresso'),
      onPressed: () async {
        String saveFilePath = '';
        saveFilePath = await saveProvider.getFilePath();
        showDialog(
            context: context,
            builder: (BuildContext context) => AlertDialog(
                title: const Text('Localização'),
                content: Text(
```

```
                    'Seu arquivo está localizado no diretório: $saveFilePath'),
                actions: <Widget>[
                  TextButton(
                    onPressed: () {
                      Navigator.pop(context, 'OK');
                    },
                    child: const Text('Ok'),
                  ),
                ],
              ));
        },
        style: Theme.of(context).textButtonTheme.style),
    ],
   ),
  )),
 ),
),
);
}
}
```

merge_sort_save_data.dart

```
class BubbleSortSaveData {
  bool isStage1Complete;
  bool isStage2Complete;
  bool isStage3Complete;

  BubbleSortSaveData(
      this.isStage1Complete, this.isStage2Complete, this.isStage3Complete);

  BubbleSortSaveData.fromJson(Map<String, dynamic> json)
      : isStage1Complete = json['isStage1Complete'],
        isStage2Complete = json['isStage2Complete'],
        isStage3Complete = json['isStage3Complete'];
```

```
  Map<String, dynamic> toJson() => {
        'isStage1Complete': isStage1Complete,
        'isStage2Complete': isStage2Complete,
        'isStage3Complete': isStage3Complete
      };
}
```

merge_sort_save_data.dart

```
class MergeSortSaveData {
  bool isStage1Complete;
  bool isStage2Complete;
  bool isStage3Complete;

  MergeSortSaveData(
      this.isStage1Complete, this.isStage2Complete, this.isStage3Complete);

  MergeSortSaveData.fromJson(Map<String, dynamic> json)
      : isStage1Complete = json['isStage1Complete'],
        isStage2Complete = json['isStage2Complete'],
        isStage3Complete = json['isStage3Complete'];

  Map<String, dynamic> toJson() => {
        'isStage1Complete': isStage1Complete,
        'isStage2Complete': isStage2Complete,
        'isStage3Complete': isStage3Complete
      };
}
```

save_data.dart

```
import 'package:sort_it_out/src/save_data/bubble_sort/bubble_sort_save_data.dart';
import 'package:sort_it_out/src/save_data/merge-sort/merge_sort_save_data.dart';

class SaveData {
  int score;
```

```
  BubbleSortSaveData bubbleSortSaveData;
  MergeSortSaveData mergeSortSaveData;

  SaveData(this.score, this.bubbleSortSaveData, this.mergeSortSaveData);

  SaveData.fromJson(Map<String, dynamic> json)
      : score = json['score'],
        bubbleSortSaveData =
            BubbleSortSaveData.fromJson(json['bubbleSortSaveData']),
        mergeSortSaveData =
            MergeSortSaveData.fromJson(json['bubbleSortSaveData']);

  Map<String, dynamic> toJson() => {
        'score': score,
        'bubbleSortSaveData': bubbleSortSaveData.toJson(),
        'mergeSortSaveData': mergeSortSaveData.toJson()
      };
}
```

save_data_provider.dart

```
import 'dart:convert';
import 'dart:developer';
import 'dart:io';

import 'package:flutter/cupertino.dart';
import 'package:path_provider/path_provider.dart';
import 'package:sort_it_out/src/save_data/bubble_sort/bubble_sort_save_data.dart';
import 'package:sort_it_out/src/save_data/merge-sort/merge_sort_save_data.dart';
import 'package:sort_it_out/src/save_data/save_data.dart';

class SaveDataProvider extends ChangeNotifier {
  SaveData? saveData;

  SaveDataProvider._create() {
    saveData = SaveData(0, BubbleSortSaveData(false, false, false),
```

```
      MergeSortSaveData(false, false, false));
}

static Future<SaveDataProvider> init() async {
  var saveDataProvider = SaveDataProvider._create();
  await saveDataProvider._init();
  return saveDataProvider;
}

_init() async {
  bool _saveFileExists = await _previousSaveFileExists();
  if (_saveFileExists) {
    _readFile();
  } else {
    _createFile();
  }
}

void _writeSaveFile() async {
  File file = File(await getFilePath());
  file.writeAsString(jsonEncode(saveData!.toJson()));
}

get bubbleSortSaveData => saveData!.bubbleSortSaveData;

get mergeSortSaveData => saveData!.mergeSortSaveData;

Future<String> getFilePath() async {
  WidgetsFlutterBinding.ensureInitialized();
  Directory appDocumentsDirectory = await getApplicationDocumentsDirectory();
  String appDocumentsPath = appDocumentsDirectory.path;
  String filePath = '$appDocumentsPath/saveFile.txt';

  return filePath;
}
```

```
void _readFile() async {
  File file = File(await getFilePath());
  String fileContent = await file.readAsString();

  try {
    saveData = SaveData.fromJson(jsonDecode(fileContent));
  } catch (e) {
    log('Error during int parsing in readFile process: $e');
  }
}

void _createFile() async {
  File file = await File(await getFilePath()).create(recursive: true);
  file.writeAsString(jsonEncode(saveData!.toJson()));
}

_previousSaveFileExists() async {
  File file = File(await getFilePath());
  return await file.exists();
}

int getSavedScore() {
  int score = saveData != null ? saveData!.score : 0;
  return score;
}

saveStageCompletion() {
  _writeSaveFile();
  notifyListeners();
}

saveScore(int score) {
  saveData!.score = score;
  _writeSaveFile();
  notifyListeners();
}
```

```
  resetSaveData() {
    saveData = SaveData(0, BubbleSortSaveData(false, false, false),
        MergeSortSaveData(false, false, false));
    _writeSaveFile();
    notifyListeners();
  }
}
```

score_system_provider.dart

```
import 'package:flutter/cupertino.dart';
import 'package:sort_it_out/src/save_data/save_data_provider.dart';

class ScoreSystemProvider extends ChangeNotifier {
  late int score;
  final SaveDataProvider? _saveDataProvider;
  ScoreSystemProvider(this._saveDataProvider) {
    if (_saveDataProvider != null) {
      int savedScore = _saveDataProvider!.getSavedScore();
      score = savedScore;
    }
  }

  _addPoints(int amount) {
    score += amount;
    _updateScoreOnSaveFile();
    notifyListeners();
  }

  _subtractPoints(int amount) {
    if (score != 0) {
      score -= amount;
      _updateScoreOnSaveFile();
    }
    notifyListeners();
```

```
  }

  resetScore() {
    score = 0;
    _updateScoreOnSaveFile();
    notifyListeners();
  }

  wrongMove() {
    _subtractPoints(50);
  }

  correctMove() {
    _addPoints(100);
  }

  hintAsked() {
    _subtractPoints(20);
  }

  void _updateScoreOnSaveFile() {
    _saveDataProvider!.saveScore(score);
  }
}
```

main.dart

```
import 'package:flutter/material.dart';
import 'package:flutter/services.dart';
import 'package:provider/provider.dart';
import 'package:sort_it_out/src/game_modes/arcade/arcade_menu.dart';
import 'package:sort_it_out/src/game_modes/arcade/stages/bubble_sort/
bubble_sort_message_provider.dart';
import 'package:sort_it_out/src/game_modes/arcade/stages/bubble_sort/stage_1/
bubble_sort_provider_1.dart';
```

```
import 'package:sort_it_out/src/game_modes/arcade/stages/bubble_sort/stage_2/
bubble_sort_provider_2.dart';
import 'package:sort_it_out/src/game_modes/arcade/stages/bubble_sort/stage_3/
bubble_sort_provider_3.dart';
import 'package:sort_it_out/src/game_modes/arcade/stages/merge_sort/
merge_sort_message_provider.dart';
import 'package:sort_it_out/src/game_modes/arcade/stages/merge_sort/stage_1/
merge_sort_provider_1.dart';
import 'package:sort_it_out/src/game_modes/arcade/stages/merge_sort/stage_2/
merge_sort_provider_2.dart';
import 'package:sort_it_out/src/game_modes/arcade/stages/merge_sort/stage_3/
merge_sort_provider_3.dart';
import 'package:sort_it_out/src/game_modes/arcade/stages/merge_sort/stage_4/
merge_sort_provider_4.dart';
import 'package:sort_it_out/src/game_modes/free/free_bubble_sort/
free_bubble_sort_provider.dart';
import 'package:sort_it_out/src/options_menu/options_menu.dart';
import 'package:sort_it_out/src/save_data/save_data_provider.dart';
import 'package:sort_it_out/src/score_system/score_system_provider.dart';
import 'theme.dart';

void main() async {
  SaveDataProvider saveProvider = await SaveDataProvider.init();
  runApp(MainMenu(saveProvider));
}

class MainMenu extends StatelessWidget {
  const MainMenu(this.saveProvider, {Key? key}) : super(key: key);
  final SaveDataProvider saveProvider;
  @override
  Widget build(BuildContext context) {
    SystemChrome.setPreferredOrientations([
      DeviceOrientation.landscapeLeft,
      DeviceOrientation.landscapeRight,
    ]);
    return MultiProvider(
```

```
      providers: [
        ChangeNotifierProvider<SaveDataProvider>(create: (_) => saveProvider),
        ChangeNotifierProvider<BubbleSortProvider1>(
            create: (_) => BubbleSortProvider1()),
        ChangeNotifierProvider<BubbleSortProvider2>(
            create: (_) => BubbleSortProvider2()),
        ChangeNotifierProvider<BubbleSortProvider3>(
            create: (_) => BubbleSortProvider3()),
        ChangeNotifierProvider<MergeSortProvider1>(
            create: (_) => MergeSortProvider1()),
        ChangeNotifierProvider<MergeSortProvider2>(
            create: (_) => MergeSortProvider2()),
        ChangeNotifierProvider<MergeSortProvider3>(
            create: (_) => MergeSortProvider3()),
        ChangeNotifierProvider<MergeSortProvider4>(
            create: (_) => MergeSortProvider4()),
        ChangeNotifierProxyProvider<SaveDataProvider, ScoreSystemProvider>(
            create: (BuildContext context) => ScoreSystemProvider(saveProvider),
            update: (context, saveProvider, scoreProvider) =>
                ScoreSystemProvider(saveProvider)),
        ChangeNotifierProvider<BubbleSortMessageProvider>(
            create: (_) => BubbleSortMessageProvider()),
        ChangeNotifierProvider<MergeSortMessageProvider>(
            create: (_) => MergeSortMessageProvider()),
        ChangeNotifierProvider<FreeBubbleSortProvider>(
            create: (_) => FreeBubbleSortProvider()),
      ],
      child: MaterialApp(
        title: 'Sort it out',
        theme: AppTheme().themeData,
        home: const HomeMenuOptions(title: 'Sort It Out'),
      ),
    );
  }
}
```

```
class HomeMenuOptions extends StatefulWidget {
  const HomeMenuOptions({Key? key, required this.title}) : super(key: key);

  final String title;

  @override
  State<HomeMenuOptions> createState() => _HomeMenuOptionsState();
}

class _HomeMenuOptionsState extends State<HomeMenuOptions> {
  @override
  Widget build(BuildContext context) {
    return Scaffold(
      appBar: AppBar(
        title: Text(widget.title),
      ),
      body: Center(
        child: Column(
          mainAxisAlignment: MainAxisAlignment.center,
          children: [
            TextButton(
                child: const Text('Jogar'),
                onPressed: () {
                  Navigator.push(
                    context,
                    MaterialPageRoute(
                        builder: (context) => const ArcadeMenu(
                            title: 'Sort It Out - Modo Arcade')),
                  );
                },
                style: Theme.of(context).textButtonTheme.style),
            // TextButton(
            //     child: const Text('Modo Livre'),
            //     onPressed: () {
            //       Navigator.push(
            //         context,
```

```
            //         MaterialPageRoute(
            //             builder: (context) =>
            //                 const FreeMenu(title: 'Sort It Out - Modo Livre')),
            //       );
            //     },
            //     style: Theme.of(context).textButtonTheme.style),
            TextButton(
              child: const Text('Opções'),
              onPressed: () {
                Navigator.push(
                  context,
                  MaterialPageRoute(
                      builder: (context) =>
                          const OptionsMenu(title: 'Sort It Out - Opções')),
                );
              },
              style: Theme.of(context).textButtonTheme.style),
          ],
        ),
      ),
    );
  }
}
```

APÊNDICE C - ARTIGO

Desenvolvimento de um Jogo para Auxílio no Ensino de Estruturas de Dados

Igor Glatz

Departamento de Informatica e Estatística Universidade Federal de Santa Catarina (UFSC) – Florianópolis, SC – Brasil

igorglatz@gmail.com

Abstract.

Data Structures are an important course in Computing related courses. Within the subject, Sorting Algorithms are a sometimes challenging concept to learn.

Having in mind the importance of the subject and the difficulties faced in learning particularly how Sorting Algorithms work, this work presents the development of a game whose objective is to facilitate the learning process of Sorting Algorithms.

After the development of the game, its quality was evaluated using the MEEGA+ method, showing positive results regarding its relevance and the students' perception of learning with the game.

Resumo.

Estruturas de Dados é uma importante disciplina em cursos relacionados a Computação. Dentro do curso, Algoritmos de Ordenação são um tema que às vezes pode ser desafiador para aprender.

Tendo em mente a importância do tema e as dificuldades encontradas ao aprender especificamente como Algoritmos de Ordenação funcionam, o presente trabalho apresenta o desenvolvimento de um jogo cujo objetivo é facilitar o processo de aaprendizagem em Algorítmos de Ordenação.

Após o desenvolvimento do jogo, sua qualidade foi avaliada utilizando o método MEEGA+, mostrando resultados positivos quanto à sua relevância e a percepção dos estudantes de aprendizado com o jogo.

1. Introdução

Com o crescente avanço tecnológico e necessidade de que todos tenham domínio pelo menos básico de conceitos de computação, não há porque não incorporar essa tecnologia no ensino desses conceitos. Os métodos mais tradicionais de aprendizagem em sala de aula, embora sejam eficientes para demonstrar conceitos abstratos e informações factuais para um grande número de alunos, não é o mais adequado quando o objetivo é obter níveis mais superiores de aprendizagem, que promovam a aplicação dos conhecimentos adquiridos em situações práticas (Choi, J. e Hannafin, M. 1995).

Assim como as tecnologias se desenvolvem com o passar do tempo, o ensino pode também se desenvolver, usufruindo dessas novas técnicas derivadas do avanço tecnológico

para a facilitação do aprendizado de conceitos e técnicas relacionadas aos cursos de Tecnologia da Informação e Computação. Como é observado por Okada e Sheehy (2020), grande parte dos alunos tem a percepção de que seria proveitoso ter diversão associada ao processo de aprendizagem, apesar de alguns alunos acharem a diversão no ensino desnecessária ou não esperada. Isso parece confirmar que os estudantes têm preferências por diferentes formas de aprendizagem, sem descartar a possibilidade de existirem problemas e dificuldades na implementação dos jogos educativos e sua incorporação no processo de aprendizagem.

De maneira geral, no ensino de computação existe um equilíbrio entre aulas práticas e teóricas, sendo essa coexistência, a base que fará com que o aluno se adapte, com visão crítica, às novas situações de sua área de formação (MEC1, 2012).

A disciplina de Estruturas de Dados, sendo em parte uma transição entre as disciplinas mais iniciais e as disciplinas mais avançadas dos cursos onde participa do currículo, tem esse papel de mostrar as ferramentas aprendidas sendo utilizadas em situações mais próximas a suas aplicações em mundo real (CHINN et al, 2003).

Além disso, considerando que as estruturas de dados em si, conceito que dá nome à disciplina, são fatores importantes no momento de arquitetura dos sistemas, principalmente quando se trata do desempenho dessas aplicações (CHINN et al, 2003), pode-se afirmar que um profissional que sabe utilizá-las corretamente possui um diferencial positivo em relação a outros profissionais que, por algum motivo, não tiveram a oportunidade de obter o mesmo conhecimento.

O pensamento computacional, compreendido como a análise e resolução de problemas por meio da construção de algoritmos (MARTINS e ELOY, 2019) vem ao encontro a essa necessidade de fornecer ao aluno situações mais reais de aplicação dos conceitos aprendidos. Ele contém princípios como reconhecimento de padrões, decomposição de problemas e abstração que ajudam a fazer essa transição de problemas com propósito mais didático a soluções mais práticas.

2. Conceitos Básicos

O pensamento computacional é compreendido como a análise e resolução de problemas por meio da construção de algoritmos (MARTINS e ELOY, 2019). Ele não é necessariamente restrito ao meio de Tecnologias de Informação, já que esses algoritmos, sendo apenas uma palavra para uma sequência de passos para executar determinada ação, podem não ser algoritmos computacionais. O termo foi cunhado por Wing (2006), que, mesmo naquela época, ressaltava a importância do pensamento computacional como habilidade para não só profissionais da área de Tecnologia da Informação, mas também para as pessoas em geral. Há um foco em resolução de problemas, reconhecimento de padrões, decomposição de problemas, projeção de sistemas, compreensão de comportamento humano e recursividade.

A disciplina de Estruturas de Dados é parte fundamental do aprendizado de Ciência de Computação e cursos relacionados, como Sistemas de Informação e Engenharia de Computação, sendo parte da categoria de Fundamentos de Desenvolvimento de Software (CC2020 TASK FORCE, 2020). A disciplina tem como objetivo geral a compreensão e construção, com o ponto de vista de Orientação a Objetos, das estruturas de dados

clássicas, como listas, filas, pilhas e árvores. Além disso, são apresentados os algoritmos de ordenação e busca relacionados a essas estruturas, assim como os ganhos/perdas de desempenho do uso de certas estruturas/algoritmos comparados com os demais (DE LUCCA, 2017).

A disciplina não somente serve de transição de solução de problemas e programação mais básica para tópicos mais avançados em Ciência da Computação, como também é a última disciplina de programação para muitos estudantes (de outros cursos como Engenharia de Computadores e Engenharia Elétrica) (CHINN et al, 2003). Há um foco em situações reais e conhecimento social acumulado pelos estudantes (CHINN et al, 2003) que torna ainda mais importante, em Estruturas de Dados, o emprego das habilidades desenvolvidas pelo Pensamento Computacional previamente citadas, como abstração e pensamento algorítmico, por exemplo.

3. Trabalhos Correlatos

Além dos trabalhos anteriormente mencionados, apresenta-se a seguir alguns trabalhos que contém conceitos aplicáveis no contexto do trabalho presente:

No trabalho de Da Rosa et al. (2020), foi desenvolvido um jogo do gênero Quebra-Cabeças em 2D, para ensino de Estruturas de Dados.

No trabalho de Dicheva e Hodge (2018) foi criado um jogo para ensinar o conceito de Pilhas (uma estrutura de dados) controlando um robô superando obstáculos.

No trabalho de Battistella et al. (2016), foi desenvolvido um jogo para a plataforma Web para o ensino do algoritmo de ordenação *Heapsort*.

No trabalho de Rosa e Coelho (2018), foi desenvolvido um jogo com uma junção de peças físicas e um aplicativo móvel para ensinar algoritmos a crianças desenvolvendo Pensamento Computacional.

Os trabalhos relacionados evidenciam a utilização de jogos, em sua maioria digitais, para o ensino de conceitos de computação. Aspectos como Pensamento Computacional, o uso de jogos para promover interatividade com algoritmos e o ensino em Estruturas de Dados são todos relevantes para o trabalho presente, com a diferença que o jogo desenvolvido tem o foco de ser utilizado em sala de aula juntamente com aulas expositivas, tirando a necessidade de que haja um tutorial extensivo dentro do jogo mas mantendo os pontos positivos da utilização de jogos no processo de aprendizado.

4. Desenvolvimento do Jogo

A partir do estudo dos trabalhos correlatos, optou-se pela utilização de uma interface interativa, por meio de *drag-and-drop*, para permitir que os alunos consigam realizar os passos dos algoritmos de busca, aumentando seu engajamento com os conceitos apresentados.

Além disso, a ideia inicial é a criação de um jogo que possa tanto ser utilizado em conjunto com aulas expositivas previamente aplicadas, quanto de maneira standalone após a aprendizagem dos conceitos, em caráter de revisão.

De forma geral o jogador tem acesso a diferentes algoritmos, que aumentam em complexidade, acompanhando a ordem de ensino observada na disciplina de Estruturas de Dados. Cada algoritmo tem uma fase correspondente, onde o aluno arrasta os elementos de alguma estrutura de dados, de acordo com o algoritmo sendo ensinado, até que complete a execução do algoritmo, tendo como resultado a estrutura de dados ordenada.

Foi feita uma avaliação de que linguagem utilizar, baseando-se em que linguagens o desenvolvedor conhecia, qual era o esforço necessário para aprender novas tecnologias e se esse esforço fazia sentido dado os limites de tempo que são naturais com esse tipo de trabalho. Foram consideradas as tecnologias *Java*, *Javascript*, *Angular* (*Typescript*), *Kotlin* ou o uso de alguma *game engine*. Como o jogo é do gênero *puzzle* e não contém elementos normalmente associados a *engines* gráficas como personagens, projéteis, cenário, inimigos, foi decidido que o foco seria nas linguagens mais tradicionais para aplicações de uso geral. Além disso, a ideia principal era que o jogo fosse graficamente leve e tivesse compatibilidade com outras plataformas (como *Web* e *iOS*). Por limites de tempo o jogo não foi testado na plataforma *iOS* e *Web*, porém há a possibilidade futura de melhorar/viabilizar a performance do jogo nelas.

Um esboço feito antes do início do desenvolvimento mostra a ideia inicial com o algoritmo *Bubble Sort,* onde dois elementos são selecionados de cada vez e o aluno deve avaliar se é necessário trocar a ordem dos elementos ou não. Seguindo a lógica do algoritmo, o menor elemento deve sempre estar localizado à esquerda. Os alunos podem também pedir dicas pagando o preço por elas em pontos.

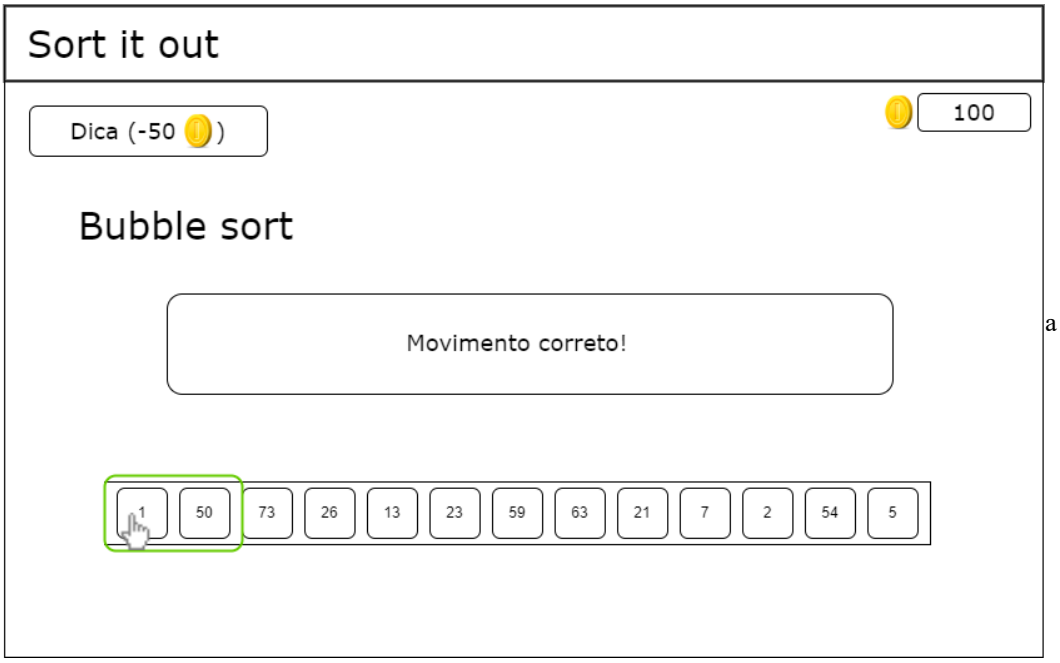


Figura 1 - Esboço inicial *Bubble Sort*

Após o desenvolvimento das fases do *Bubble Sort,* incrementando a dificuldade com cada iteração, foram implementadas também fases para o algoritmo *Merge Sort,* que

possui a estratégia de "divisão e conquista" onde a estrutura a ser ordenada é dividida em pedaços menores até os elementos individuais, depois aplicando uma lógica de *merge* para unir os elementos novamente, porém de forma ordenada. No algoritmo *Merge Sort*, foi decidido criar dois tipos diferentes de fase:

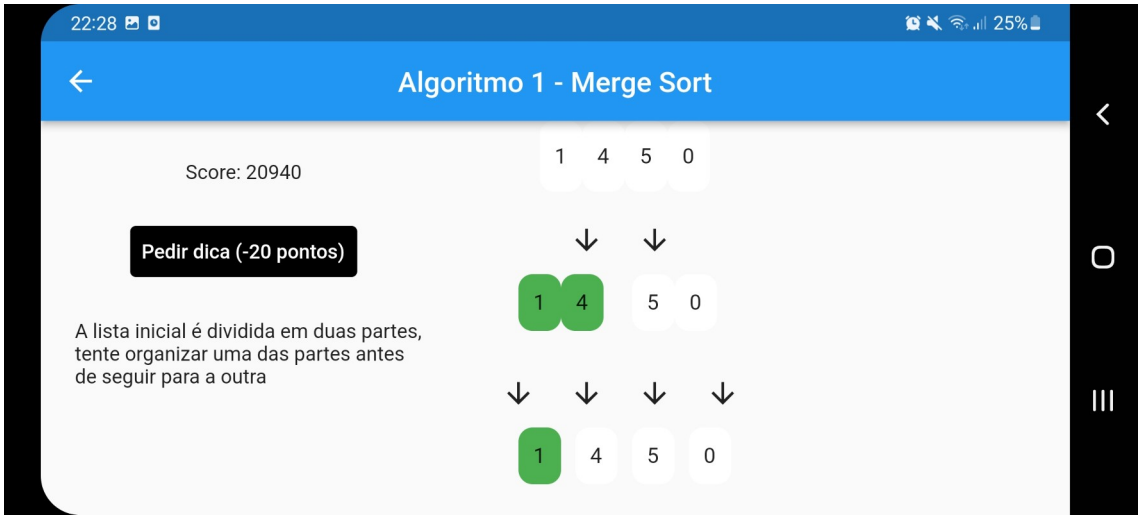


Figura 2 - Tela da primeira fase do *Merge Sort*

Nesse primeiro tipo de fase o jogador apenas indica através de toques nas setas, a ordem na qual o algoritmo *Merge Sort* executa os passos já predispostos na tela. Cada passo escolhido de maneira errada diminui os pontos do aluno, assim como cada acerto faz o aluno ganhar pontos. O botão de pedir dicas ainda é presente.

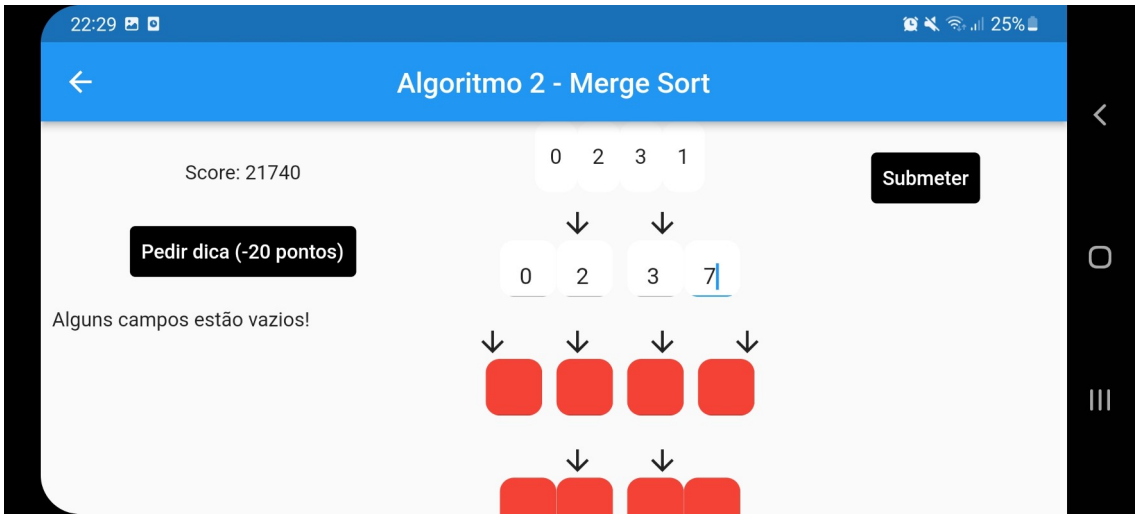


Figura 3 - Tela da segunda fase do *Merge Sort*

Já no segundo tipo de fase, o aluno deve preencher a estrutura montada com os elementos corretos em cada posição dos passos do algoritmo. Após preencher todos os espaços o aluno deve apertar o botão "Submeter" para confirmar sua solução. Soluções erradas fazem o aluno perder pontos, e a solução correta faz o aluno ganhar pontos.

5. Avaliação

Após o desenvolvimento do jogo, foi realizada uma avaliação de qualidade do jogo utilizando o método MEEGA+ de avaliação de qualidade de jogos educacionais em termos de usabilidade e experiência do jogador, da perspectiva do jogador no contexto de educação em computação (PETRI et al., 2019).

O modelo se baseia na medição de certas dimensões para avaliar a qualidade do software criado, sendo essas dimensões divididas em duas categorias: Usabilidade e Experiência de Jogador. Na categoria de Usabilidade, se encontram as dimensões de Estética, Capacidade de aprendizado, Operabilidade e Acessibilidade. Já na categoria de Experiência de Jogador, são englobadas as dimensões de Proteção de Erro do Usuário, Atenção Focada, Diversão, Desafio, Interação Social, Confiança, Relevância, Satisfação e Aprendizagem Percebida.

A execução da avaliação foi realizada em sala de aula, na disciplina de Estruturas de Dados, no curso de Sistemas de Informação, na Universidade Federal de Santa Catarina. Os participantes foram 15 alunos da disciplina, todos do sexo masculino e com idade entre 18 e 29 anos. Foi feita uma apresentação simples com poucos slides explicando superficialmente o objetivo de cada fase, além da motivação inicial para a criação do aplicativo e características do seu desenvolvimento. Também foi dada uma breve introdução sobre o que é o MEEGA+.

A aplicação do jogo aconteceu perto do final do primeiro semestre letivo de 2023, o que significa que os alunos já tinham conhecido os algoritmos presentes no jogo, agilizando a explicação do aplicativo. A premissa inicial era explicar os algoritmos pela primeira vez logo antes de aplicar o jogo, porém foi aproveitada essa oportunidade para avaliar como o jogo se comporta em nível de revisão de conceitos aprendidos a um tempo atrás.

O tempo de aula previsto era de 90 minutos, com a apresentação inicial sendo de mais ou menos dez minutos e um tempo de em torno de 15 a 20 minutos para preparar o uso do aplicativo nos celulares dos alunos. O aplicativo foi distribuído pelo Moodle da disciplina, onde os alunos baixaram o pacote Android (APK) da aplicação e o instalaram em seus dispositivos. Após o uso do aplicativo por em torno de 30 minutos, os alunos responderam ao Questionário do Aluno em formato de um formulário Google (Google Forms), com todas as questões do questionário original reproduzidas digitalmente, também disponibilizado aos alunos pelo Moodle da disciplina.

Para gerar tabelas e gráficos dos resultados da avaliação, foi utilizada a planilha disponibilizada juntamente ao Questionário do Aluno do MEEGA+, inserindo os dados na planilha de acordo com as instruções também disponibilizadas no site do GQS (http://www.gqs.ufsc.br/quality-evaluation/meega-plus/) grupo ao qual os autores do MEEGA+ pertencem (PETRI et al., 2019).

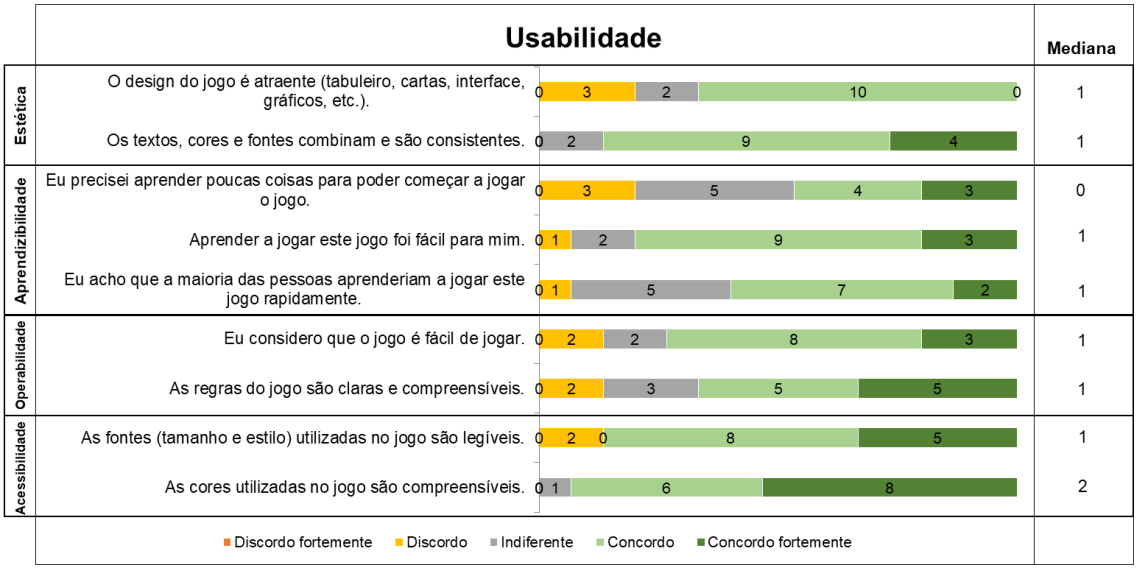


Figura 4 - Gráfico de Usabilidade

Observando a Figura 4, vemos que os alunos de maneira geral responderam de forma positiva às questões de Usabilidade em relação ao jogo, com destaque positivo para a utilização de cores, fontes, textos e facilidade de aprender a jogar. Nos destaques negativos, as proposições "Eu precisei aprender poucas coisas para poder começar a jogar o jogo" e "Eu acho que a maioria das pessoas aprenderiam a jogar esse jogo rapidamente" foram as com menor conceito. Isso se deve provavelmente à apresentação do aplicativo ser muito breve, talvez por receio de entregar informações demais e tornar o jogo muito fácil de ser resolvido. Uma apresentação com instruções mais detalhadas do funcionamento do jogo possivelmente ajudaria nesse sentido. Além disso, uma observação recorrente foi que o fluxo de resolver uma fase e ir para a seguinte não tinha clareza o suficiente, alguns alunos relataram inclusive completar a mesma fase repetidas vezes sem necessidade, visto que o jogo não redireciona o aluno para a fase seguinte ao completar uma fase. Fazer esse redirecionamento provavelmente sanaria essa deficiência do jogo.

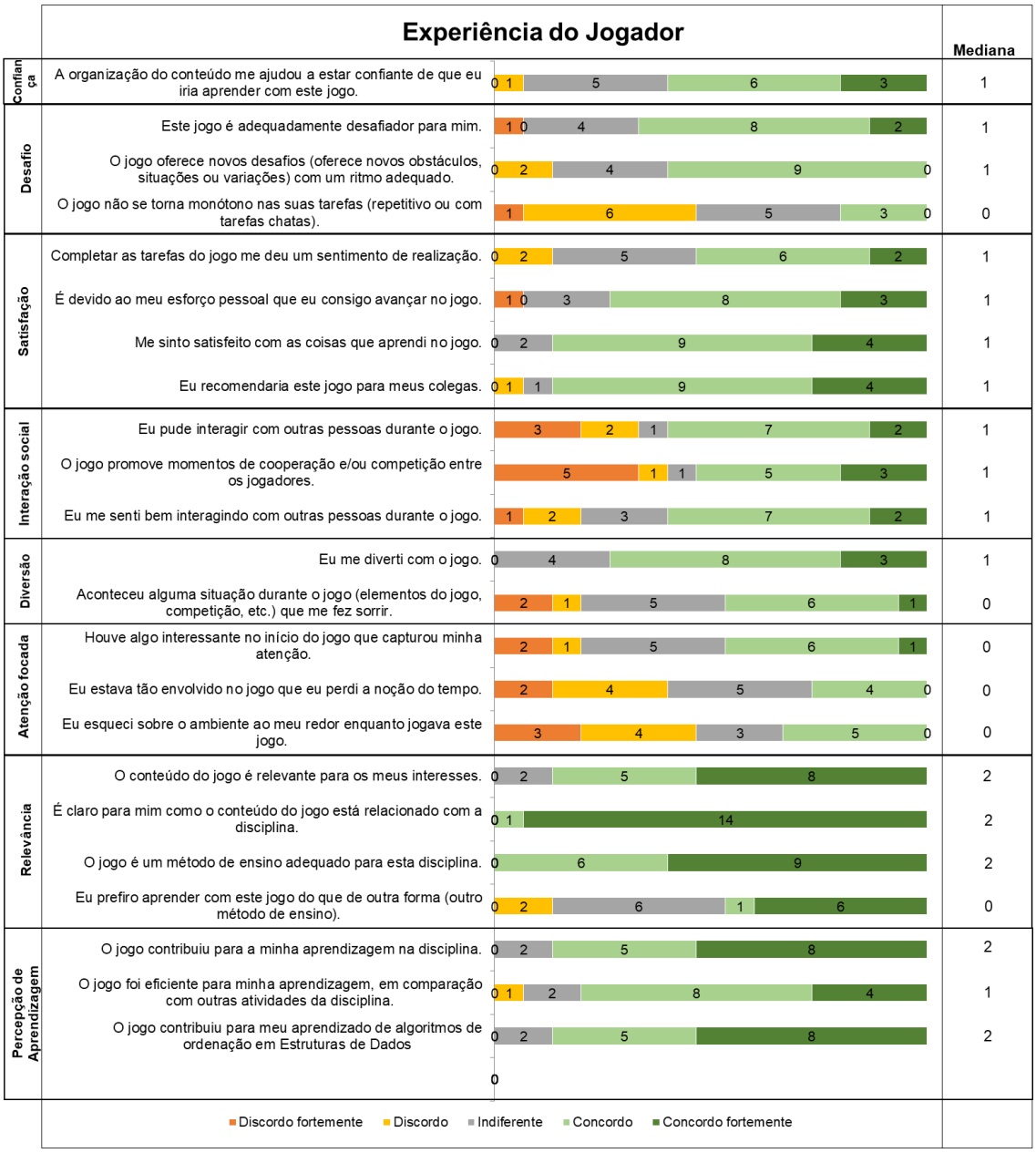


Figura 6 - Gráfico de Experiência de Usuário

Quanto à experiência do jogador, os pontos positivos são a percepção de aprendizagem e a relevância, os alunos relataram que a interação com os algoritmos os fez perceber o quão ineficiente o *Bubble Sort* é, por exemplo, com suas repetidas voltas pela estrutura a ser ordenada. Como pontos negativos, a interação social, se tratando de um jogo individual é compreensivelmente baixa. Além disso os jogadores não se sentiram imersos enquanto jogavam e notaram monotonia nas funções do jogo. Devido à natureza repetitiva de algoritmos de ordenação talvez a estratégia de fazer os alunos executarem todos os passos um a um, principalmente no caso do Bubble Sort tenha causado esse problema.

6. Conclusão

O objetivo principal desse trabalho foi criar um jogo que pode ser utilizado como ferramenta para auxílio na aprendizagem de algoritmos de ordenação na disciplina de Estruturas de Dados.

O desenvolvimento do aplicativo foi repleto de desafios e acabou sendo mais complexo do que tinha sido pensado inicialmente. A barreira de criar algoritmos interativos foi algo difícil de superar, a tecnologia Flutter tem algumas limitações em relação a aplicações dessa forma. Talvez um estudo mais amplo de tecnologias diferentes pudesse revelar uma ferramenta mais adequada para o desenvolvimento de um aplicativo dessa categoria.

Durante a pesquisa de trabalhos correlatos a ferramenta MEEGA+ se destacou como uma das mais completas em avaliação da qualidade de jogos educacionais, o que incentivou o seu uso para a avaliação do aplicativo Sort it Out.

Ao pôr em prática a avaliação, utilizando o MEEGA+, foram levantados pontos fortes e fracos da aplicação. Entre os destaques positivos se encontram a relevância do aplicativo em relação ao conteúdo aprendido e a percepção de aprendizagem que os alunos tiveram. Já como pontos fracos, a experiência de usuário poderia ter sido mais desenvolvida, com explicações mais detalhadas do funcionamento do jogo, além de alterações menores na interface para melhorar a usabilidade do jogo de forma geral.

Por fim, como melhorias futuras se sugere a adição de mais algoritmos diversificados, com fases possuindo desafios também diversificados, a fim de aumentar o repertório do aplicativo e oferecer uma experiência mais completa dos algoritmos de ordenação em Estruturas de Dados.

Referências

Choi, J. e Hannafin, M. 1995. **Situated Cognition and Learning Environments: Roles, Structures and Implications for Design**. Educational Technology Research and Development, 43(2), p. 53-69.

Ministério da Educação. 2012. **Diretrizes Curriculares Nacionais para Cursos de Graduação em Computação**. Parecer CNE/CNS 136/2012. Disponível em: <http:// portal.mec.gov.br>. Acesso em: 24 fevereiro, 2022.

CHINN, Donald; PRINS, Phil; TENENBERG, Josh. **The Role Of The Data Structures Course In The Computing Curriculum.** Journal of Computing Sciences in Colleges, Evansville, v. 19, n. 2, p. 91-93, 2003.

CC2020 TASK FORCE. **Computing Curricula 2020: Paradigms for Global Computing Education**. Nova Iorque: Association for Computing Machinery, 2020. 205 p. ISBN 978-1-4503-9059-0. DOI https://doi.org/10.1145/3467967. Disponível em: https://dl.acm.org/doi/book/10.1145/3467967. Acesso em: 15 fev. 2022.

DE LUCCA, José Eduardo. **Plano de Ensino INE5609 - Estruturas de dados**. Universidade Federal de Santa Catarina. Centro Tecnológico. Departamento de Informática e Estatística, Florianópolis, 14 jul. 2017.

MARTINS, Amilton Rodrigo de Quadros; ELOY, Adelmo Antonio da Silva. **EDUCAÇÃO INTEGRAL POR MEIO DO PENSAMENTO COMPUTACIONAL: letramento em programação**: relatos de experiência e artigos científicos. Curitiba: Appris, 2019. 363 p.

WING, J. **PENSAMENTO COMPUTACIONAL** Um conjunto de atitudes e habilidades que todos, não só cientistas da computação, ficaram ansiosos para aprender e usar. Revista Brasileira de Ensino de Ciência e Tecnologia , v. 9, n. 2, 2016. Disponível em: http://dx.doi.org/10.3895/rbect.v9n2.4711.

DA ROSA, Alexandre; DE JESUS, Andreia; IGARASHI, Gabriel Vaz ; PEREIRA, Vinicius Struginski . I**ron Ears: primeiras impressões de um jogo educativo para ensino de estrutura de dados lineares.** In: WORKSHOP SOBRE EDUCAÇÃO EM COMPUTAÇÃO (WEI), 28 , 2020, Cuiabá. Anais [...]. Porto Alegre: Sociedade Brasileira de Computação, 2020 . p. 31-35. ISSN 2595-6175. DOI: https://doi.org/10.5753/ wei.2020.11124.

ROSA, Yuri Kaiser da; COELHO, Everton Schafaschek. **Space Code: Um Jogo Para O Desenvolvimento De Pensamento Computacional Utilizando Interfaces Tangíveis**. 2018. 151 f. TCC (Graduação) - Curso de Sistemas de Informação, Centro Tecnológico, Universidade Federal de Santa Catarina, Florianópolis, 2018. Cap. 1.

BATTISTELLA, Paulo E. et al. SORTIA 2.0: A **sorting game for data structure teaching**. In: BRAZILIAN SYMPOSIUM ON INFORMATION SYSTEMS, 12., 2016, Porto Alegre. Proceedings of the XII Brazilian Symposium on Information Systems. Porto Alegre: Brazilian Computer Society, 2016. p. 558-565.

BATTISTELLA, Paulo E. et al. **SORTIA 2.0: A sorting game for data structure teaching**. In: BRAZILIAN SYMPOSIUM ON INFORMATION SYSTEMS, 12., 2016,

Porto Alegre. Proceedings of the XII Brazilian Symposium on Information Systems. Porto Alegre: Brazilian Computer Society, 2016. p. 558-565.

DICHEVA, Darina; HODGE, Austin. A**ctive Learning through Game Play in a Data Structures Course**. In: ACM TECHNICAL SYMPOSIUM ON COMPUTER SCIENCE EDUCATION, 49., 2018, Nova Iorque. Proceedings of the 49th ACM Technical Symposium on Computer Science Education (SIGCSE '18). Nova Iorque: Association For Computing Machinery, 2018. p. 834-839.

PETRI, Giani; WANGENHEIM, Christiane Gresse von; BORGATTO, Adriano Ferreti. **MEEGA+: Um Modelo para a Avaliação de Jogos Educacionais para o ensino de Computação**. Revista Brasileira de Informática na Educação, [S.l.], v. 27, n. 03, p. 52-81, dez. 2019. ISSN 2317-6121. Disponível em: <https://br-ie.org/pub/index.php/rbie/article/view/v27n035281>. Acesso em: 02 mar. 2022. doi:http://dx.doi.org/10.5753/rbie.2019.27.03.52.