UNIVERSIDADE FEDERAL DO CEARÁ

CAMPUS DE QUIXADÁ

CURSO DE GRADUAÇÃO EM ENGENHARIA DE SOFTWARE

ROBERTSON DA SILVA NASCIMENTO

O USO DA FERRAMENTA APP INVENTOR NO ENSINO DE PROGRAMAÇÃO EM CURSOS DE TI

QUIXADÁ

2023

ROBERTSON DA SILVA NASCIMENTO

O USO DA FERRAMENTA APP INVENTOR NO ENSINO DE PROGRAMAÇÃO EM CURSOS DE TI

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software do Campus de Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Orientador: Prof. Dr. Jefferson de Carvalho Silva.

QUIXADÁ

2023

Dados Internacionais de Catalogação na Publicação
Universidade Federal do Ceará
Sistema de Bibliotecas
Gerada automaticamente pelo módulo Catalog, mediante os dados fornecidos pelo(a) autor(a)

N198u Nascimento, Robertson da Silva.
O uso da ferramenta App Inventor no ensino de programação em cursos de TI / Robertson da Silva Nascimento. – 2023.
76 f. : il. color.

Trabalho de Conclusão de Curso (graduação) – Universidade Federal do Ceará, Campus de Quixadá, Curso de Engenharia de Software, Quixadá, 2023.
Orientação: Prof. Dr. Jefferson de Carvalho Silva.

1. Ensino. 2. Aprendizagem. 3. Programação. 4. Aplicativos Móveis. I. Título.

CDD 005.1

ROBERTSON DA SILVA NASCIMENTO

O USO DA FERRAMENTA APP INVENTOR NO ENSINO DE PROGRAMAÇÃO EM CURSOS DE TI

Trabalho de Conclusão de Curso apresentado ao Curso de Graduação em Engenharia de Software do Campus de Quixadá da Universidade Federal do Ceará, como requisito parcial à obtenção do grau de bacharel em Engenharia de Software.

Aprovada em: ____/____/____.

BANCA EXAMINADORA

Prof. Dr. Jefferson de Carvalho Silva (Orientador)
Universidade Federal do Ceará (UFC)

Prof. Dr. Emanuel Ferreira Coutinho
Universidade Federal do Ceará (UFC)

Prof. Dr. Wladimir Araujo Tavares
Universidade Federal do Ceará (UFC)

Dedicado ao meu pai, Pedro Bernardino Nascimento (vulgo Pedro Pajeú), que se foi em 2020 vítima de tétano. Ele me ensinou que simplicidade não se trata de *ter* mas sim de *ser*. Pai, essa é pra você.

AGRADECIMENTOS

Ao Prof. Dr. Jefferson de Carvalho, pela excelente orientação, sempre prático e direto ao ponto me guiou da melhor forma até o final do projeto.

Aos professores participantes da banca examinadora Prof. Dr. Emanuel Ferreira Coutinho e Prof. Dr. Wladimir Araujo Tavares pelo tempo, pelas valiosas colaborações e sugestões. Não tive a sorte de ter aula com o professora Emanuel, mas o agradeço de qualquer forma. Fiz Matemática Discreta e Linguagens de Programação com o professor Wladimir, obrigado por não desistir de mim professor.

Ao Prof. Dr. David Sena Oliveira, meu professor de Fundamentos de Programação, por me contagiar com seu entusiasmo por programar.

Aos colegas de universidade, pelas reflexões, críticas e sugestões recebidas. Em especial aos meus amigos Bergson e Gabriel. Sem esquecer dos participantes da pesquisa que possibilitaram o desenrolar deste trabalho, estamos juntos pessoal.

A minha mãe Maria da Silva Nascimento (dona Aldeide), que sempre me apoiou, tanto financeiramente, quanto emocionalmente, em meus estudos. Sendo além de mãe, a minha melhor amiga.

Ao meu irmão Dr. Jose Roberto (Betim), que sempre me inspirou por usa disciplina e dedicação em sua carreira. Betim macho tu ainda me deve aquele coop de Resident Evil 5.

Ao meu irmão Júnior, que me ensinou a ter maturidade e resolver problemas cotidianos com naturalidade. Também a ter bom gosto para filmes, tipo do diretor Quentin Tarantino, e interesse por Filosofia.

Aos meus amigos conterrâneos Alequim e Sidcley, que quase todo final de semana se reunimos para jogar aquela conversa fora e tomar uma cervejinha. Ninguém é de ferro.

A minha esposa, que desde 2018 me aguenta. Te amo Camila.

"Não ensino meus alunos. Crio a condição para que aprendam." (Albert Einstein)

RESUMO

Com o aumento da procura por ingresso em cursos de graduação em Tecnlogia de Informação (TI), o cenário atual do ensino de TI enfrenta alguns desafios. Um deles é a desigualdade entre o nível de conhecimento de programação que os estudantes possuem ao chegarem na universidade. Pelo fato de haver estudantes que já possuem experiência com desenvolvimento de software, até mesmo com interfaces gráficas, enquanto outros ainda nem mesmo conhecem a definição de um algoritmo, podem ocorrer desestímulos precoces neste segundo grupo. Desta forma, partindo de que se faz necessário o uso de meios alternativos de ensino de programação, que estimulem mais os estudantes, por possibilitar a criação de aplicativos móveis mais parecidos com os do mercado atual, este trabalho realizou um experimente com um grupo de estudantes de graduação da área de TI. Usando a ferramenta de programação por blocos App Inventor foram então criadas videoaulas, e um material de estudos auxiliar, sendo estes consumidos por um grupo de alunos. Este grupo, juntamente com um segundo grupo que não consumiu tais materiais, foram então submetidos a realização de testes de julgamento sobre o uso programação por blocos, e habilidade de programação. Sendo feita, pelos resultados colhidos destes testes, uma comparação e análise entre os dois grupos. Após esta análise foram concluídos resultados parecidos entre os grupos, porém, com o grupo consumidor dos materiais se saindo um pouco melhor.

Palavras-chave: Ensino; Aprendizagem; App Inventor; Programação por Blocos; Aplicativos Móveis.

ABSTRACT

With the increase in demand for admission to undergraduate courses in Information Technology (IT), the current scenario of IT education faces some challenges. One of them is the inequality between the level of programming knowledge that students have when they arrive at university. Due to the fact that there are students who already have experience with software development, even with graphical interfaces, while others do not even know the definition of an algorithm, early discouragement may occur in this second group. In this way, based on the fact that it is necessary to use alternative means of teaching programming, which stimulate students more, by enabling the creation of mobile applications that are more similar to those in the current market, this work carried out an experiment with a group of students degree in the IT field. Using the App Inventor block programming tool, video classes were then created, as well as auxiliary study material, which were consumed by a group of students. This group, along with a second group that did not consume such materials, were then subjected to judgment tests on block programming use, and programming ability. Based on the results collected from these tests, a comparison and analysis between the two groups was carried out. After this analysis, similar results were concluded between the groups, however, with the group consuming the materials doing a little better.

Keywords: Teaching; Learning; App Inventor; Block Programming; Mobile Applications.

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ABREVIATURAS E SIGLAS

FUP	Fundamentos de Programação
IDE	*Integrated Development Environment*
MIT	*Massachusetts Institute of Technology*
SDK	*Software Development Kit*
SO	Sistema Operacional
TI	Tecnlogia de Informação
UFC	Universidade Federal do Ceará
UI	*User Interface*
USP	Universidade de São Paulo

SUMÁRIO

1 INTRODUÇÃO

O interesse pelo estudo de Computação tem crescido cada vez mais com o avanço da informatização e inclusão digital na sociedade. Como afirma Camargo (2021) “O momento atual requer propostas tecnológicas inovadoras e a área da computação é essencial para a transformação digital tão presente neste momento”. Também é importante citar o fator de possibilidade do trabalho remoto na área de computação evidenciado pela pandemia do COVID 19. Concomitante a isso, segundo o site GeekHunter - plataforma de recrutamento de profissionais de TI, apenas no ano de 2020 houve um aumento de 310% no número de vagas no mercado de TI (GIORDAN, 2021). Logo, a busca por ingresso em cursos de graduação relacionados à Computação tem aumentado cada vez mais.

Um dos primeiros desafios que os estudantes enfrentam quando ingressam em uma graduação de computação é a aprovação nas disciplinas de introdução à programação. Nelas são apresentados os conceitos básicos de programação. Dentre eles podemos citar: conceito de algoritmos, método para construção de algoritmos, tipos de algoritmos, variáveis, estruturas de controle e repetição e funções (ASCENCIO; CAMPOS, 2012). Ou seja, são disciplinas que apresentam os conceitos fundamentais para o bom desempenho dos alunos no restante dos seus cursos.

Contudo, as circunstâncias em que cada aluno começa a graduação são diversas, tendo como principal diferencial os alunos que cursaram ensino médio profissionalizante ou técnico, e entram na universidade já tendo uma boa base de lógica de programação, conceitos de estrutura de dados, e até conhecimento em desenvolvimento Web ou *Mobile*. Já alunos que cursaram ensino médio regular, ou que, em geral, nunca tiveram contato com programação antes da universidade, apresentam maior dificuldade nesse primeiro contato. Isso leva a evasões, desistências, e desestímulos por partes dos alunos no início de seus cursos.

No que diz respeito ao aprendizado em Computação no início da graduação, vem à tona a necessidade dos estudantes possuírem uma base de conhecimentos anteriores que servirá como alicerce para o devido aproveitamento dos ensinos que lhes são passados dentro da universidade. Nesse contexto, se torna indispensável o conhecimento dos docentes sobre os déficits que estudantes não oriundos de escolas profissionais possuem nesse período inicial da graduação.

Estando então presente esta dificuldade em aprendizado de programação nos semestres iniciais, principalmente por alunos de ensino médio não profissionalizante. Além disso, o

ambiente de programação inicial (terminal/console) acaba não gerando uma saída empolgante aos olhos dos alunos. Logo, com a finalidade de mitigar este problema, a ferramenta **App Inventor** vem como um complemento no processo de ensino, unindo os conceitos básicos de programação, numa forma amigável e gerando uma saída mais semelhante a softwares do mercado: uma aplicação móvel.

O App Inventor é uma plataforma de desenvolvimento de aplicativos para dispositivos móveis, através de programação por blocos. Além disso, por ser uma plataforma Web, não há necessidade de instalação de nenhum software no computador do estudante. Tendo o ciclo de vida de desenvolvimento muito rápido, considerando desde o primeiro acesso à plataforma, até o passo de execução do aplicativo criado, no celular do estudante (coisa que praticamente todo jovem possui). Assim, a ferramenta se torna um ambiente propício tanto para o ensino de programação, quanto no fator gerador de estímulo de estudo de Computação, causado nos alunos.

Em relação às vantagens de uso da programação por blocos, podemos ver algumas pelo trecho:

> "Atualmente, a utilização do paradigma de Programação em Blocos (PB), vem apresentando resultados positivos no ensino do pensamento computacional. Esse paradigma vem sendo adotado por várias metodologias, desde programação de histórias, jogos, vídeos até a programação de robôs"(SOUSA *et al.*, 2020, p. 1514).

Os trabalhos relacionados Guimarães *et al.* (2016) e Vieira *et al.* (2020) mostram exemplos do uso do App Inventor para o ensino de programação. Estes trabalhos obtiveram conclusões que corroboraram com incentivo ao uso de tal ferramenta no ambiente acadêmico.

Os motivos pelos quais o App Inventor foi a ferramenta de programação por blocos escolhida por este projeto são: (i) a maturidade da plataforma, tendo sido lançado em dezembro de 2010; (ii) é desenvolvido e continuado pelo *Massachusetts Institute of Technology* (MIT), tendo desta forma um bom respaldo de documentação; (iii) possui maior acervo de trabalhos acadêmicos relacionados publicados.

1.1 Objetivos

Os objetivos deste trabalho são listados nessa seção.

1.1.1 Objetivos Gerais

Avaliar o uso da ferramenta App Inventor como um meio lúdico do ensino de Programação em cursos de Computação.

1.1.2 Objetivos Específicos

- Evidenciar a necessidade da geração de estímulos em estudantes de semestres iniciais de cursos de graduação, que nunca tiveram contato com programação.
- Desenvolver material on-line com o intuito de guiar o aluno no ensino de programação.
- Avaliar o impacto da utilização da ferramenta App Inventor no ensino de programação.

1.2 Organização do Texto

Este trabalho se organiza falando um pouco sobre conceitos tratados no contexto de dispositivos móveis, desenvolvimento *Mobile*, App Inventor, e ensino de programação no Capítulo 2. No Capítulo 3 são listados os trabalhos relacionados, sendo os dois primeiros sobre a utilização do App Inventor como ferramenta de ensino de programação, o terceiro mais voltado a listar as principais dificuldades no processo de aprendizagem de programação, e o quarto sobre o ensino de programação com uma ferramenta diferente de programação por blocos, o Scratch. Já no Capítulo 4 é apresentada a metodologia que será seguida para fins de alcançar os objetivos gerais e específicos deste trabalho.

2 FUNDAMENTAÇÃO TEÓRICA

Este capítulo fornece a base de fundamentação teórica para o entendimento do corrente trabalho. De forma que os principais conceitos foram separados em seções.

2.1 Sistemas Operacionais

Um dos principais aspectos que estão presentes em aplicações *Mobile*, são praticidade no uso cotidiano, facilidade de acesso, e, quando comparado com computadores, menor preço. Inerentemente a estes aplicativos, estão os SOs sob os quais eles funcionam, e os *frameworks* nos quais são desenvolvidos.

Os primeiros computadores feitos tinham tamanhos de salas de escritórios, para programar em um computador desse tipo, era preciso mudar manualmente seu hardware, isso podeira demorar bastante, às vezes, mais do que o aceitável. Com a evolução da Computação, em especial, da Engenharia de Computação, os SOs surgiram como uma forma de intermediar o gerenciamento de tarefas genéricas do computador, para a utilização do mesmo por um usuário comum. O SO, então, fornece os meios corretos de gerenciamento de recursos computacionais para a realização das operações requisitadas pelo usuário do computador. Portanto, um SO funciona de forma simular a um governo. Assim como um governo, o SO não tem uma função útil por se só, ele apenas fornece o ambiente correto nos quais os programas irão realizar o trabalho útil (SILBERSCHATZ *et al.*, 2018).

2.1.1 Android

Atualmente o mundo *Mobile* é dividido em dois SOs, são eles: Android e iOS, onde os mesmos possuem suas semelhanças e particularidades. O Android foi oficialmente lançado em 2003, baseado no SO *open source* Linux (FAUSTINO *et al.*, 2017).

Foi em Polo Alto, na Califórnia, onde Andy Rubin, Rich Miner, Nick Sears e Chris White (mais detalhes na Tabela 1) fundaram a empresa Android Inc, esta que em 2005 foi comprada, com sua equipe de desenvolvedores, pela Google. Inicialmente eles tinham foco em câmeras, porém com o mercado em baixa, migraram para smartphones. O sistema surgiu com a ideia de tornar o entendimento de fácil acesso para programadores. “Seguindo a ideologia de Andy Rubin, de um sistema totalmente gratuito e de fácil entendimento, o Google decidiu liberar parte do seu código sob a Licença Apache de código de aberto” (FAUSTINO *et al.*, 2017,

p. 101), isso acabou funcionando como jogada de marketing proporcionando a aderência de grandes companhias a utilização do Android em seus aparelhos, como, por exemplo, a Motorola e Samsung, facilitando o processo de desenvolvimento de aplicativos, e expansão de mercado do SO.

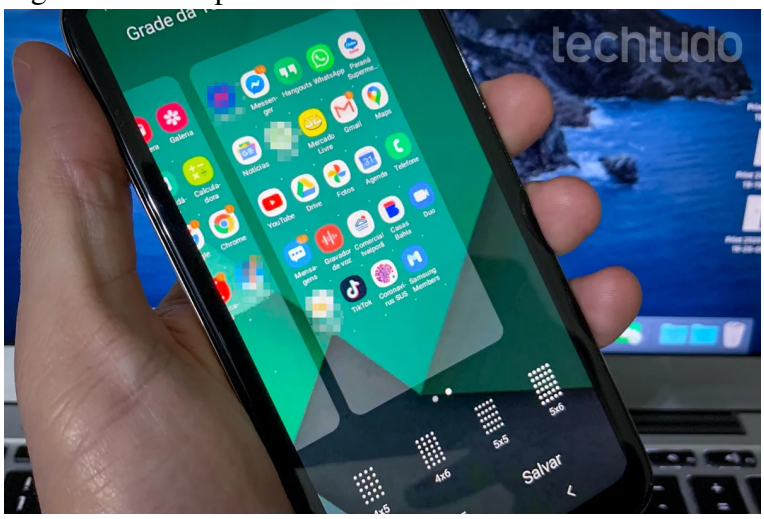
Tabela 1 – Criadores do Android

Nome	**Formação**	**Profissão**
Andy Rubin	Graduado em Ciência da Computação - Utica University	Programador
Rich Miner	Doutorado em Ciência da Computação - University of Massachusetts Lowell	Cientista da Computação
Nick Sears	Mestrado em Administração de Empresas - Pacific Lutheran University	Administrador de Empresas
Chris White	Graduado em Ciência da Computação - Syracuse University	Engenheiro de Software

Fonte: Elaborado pelo autor.

Pelo fato do Android ser baseado na utilização do *kernel* do Linux, o atributo de segurança ganha força no SO, visto, por exemplo, pela citação "manter a segurança de um sistema Linux é fácil de implementar e isso protege a integridade das informações confidenciais dentro do sistema" (YASWINSKI *et al.*, 2019). Outra característica predominante em plataformas Android é que por ser *open source*, a sua capacidade de ser implementada e personalizada por vários fabricantes diferentes, o torna um SO flexível e altamente compatível com múltiplos dispositivos. A figura 1 mostra um exemplo de interface gráfica do SO Android.

Figura 1 – Exemplo de interface do SO Android



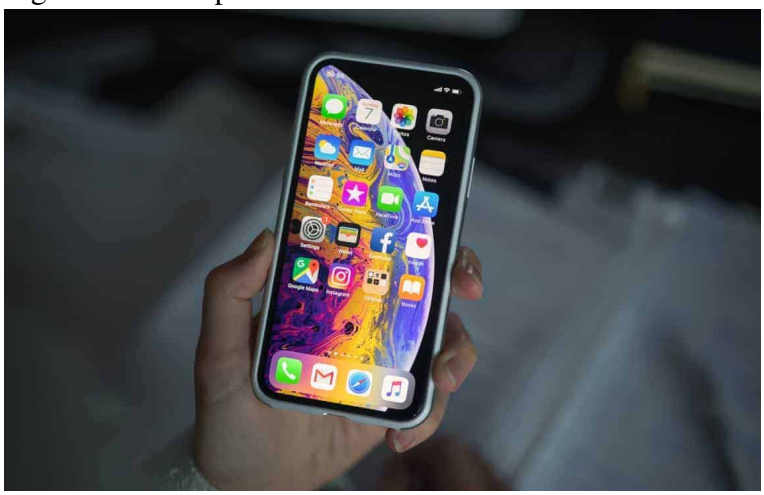
Fonte: https://www.techtudo.com.br/dicas-e-tutoriais/2020/08/como-diminuir-o-tamanho-dos-aplicativos-na-tela-do-celular.ghtml

2.1.2 iOS

Diferentemente do Android, o iOS é monopólio da empresa Apple, ou seja, para desenvolver nativamente para iOS, é preciso ter um computador Apple. Por causa disso, os usuários ficam presos a um mesmo fabricante (FAUSTINO *et al.*, 2017).

O SO iOS foi lançado juntamente com o primeiro iPhone, baseado no Mac OS X, funciona como uma versão miniatura do SO da Apple presente nos seus *desktops*, a Tabela 2 traz detalhes sobre os principais envolvidos no projeto. Sua principal característica é o foco em otimização no relacionamento entre software e hardware, também pelo fato de ambos serem monopólio da mesma companhia. O que é demasiado desvantajoso quando se trata de iPhones, é seu alto preço, diminuindo assim a quantidade de usuários da plataforma em países que não são considerados de primeiro mundo. A figura 2 mostra um exemplo de interface gráfica do SO iOS.

Figura 2 – Exemplo de interface do SO iOS



Fonte: https://www.olhardigital.com.br/2020/01/22/dicas-e-tutoriais/aprenda-a-colocar-imagens-nos-icones-dos-aplicativos-no-ios/

Tabela 2 – Principais envolvidos no lançamento do primeiro iOS

Nome	**Formação**	**Profissão**
Scott Forstall	Mestre em Ciência da Computação - Universidade de Stanford	Engenheiro de Software
Steve Jobs	Não graduado	CEO da Apple

Fonte: Elaborado pelo autor.

2.2 Tecnologias de Desenvolvimento

Partimos então para as tecnologias de desenvolvimento de aplicativos móveis. O assunto de desenvolvimento *Mobile* pode até ser definido como uma moda atualmente, logo, existem diversos *frameworks* voltados para esse meio.

Ao falar de desenvolvimento *Mobile*, uma das principais características é sua separação em três tipos distintos: nativo, híbrido, e web apps.

2.2.1 Nativo

De acordo com Verma *et al.* (2018) "Um aplicativo móvel nativo é um aplicativo que apenas funciona em um sistema operacional específico usando seu *Integrated Development Environment* (IDE) e *Software Development Kit* (SDK).". Em outras palavras, se um aplicativo foi desenvolvido nativamente para funcionar no SO Android, ele apenas funcionará em smartphones Android, o mesmo vale para aplicativos nativos iOS.

Ao programar utilizando a IDE e SDK específico de uma plataforma de desenvolvimento nativo, o programador acessa diretamente os componentes nativos oferecidos pela respectiva plataforma, assim, manipulando-os no decorrer da construção de um aplicativo. As linguagens de programação de desenvolvimento nativo são Kotlin e Java para Android, Swift e Objective-C para iOS. Desta forma, se impossibilita o funcionamento de um aplicativo nativo Android em um sistema iOS, ou vice-versa.

É importante ressaltar que aplicativos desenvolvidos nativamente, são projetados também se pensando em obter maior desemprenho na relação hardware-software. Esse aspecto, no que lhe concerne, pode ser considerado uma vantagem do modelo de desenvolvimento nativo (VERMA *et al.*, 2018).

2.2.2 Híbrido

Um aplicativo híbrido tem como principal diferencial, ser desenvolvendo em um framework que possibilita seu funcionamento em ambos os SOs *Mobile* (Android e iOS). Fica claro então sua maior vantagem sobre aplicativos desenvolvidos nativamente: código multiplataforma, onde só é necessário codificar uma vez a aplicação e pronto, ela já funciona nos dois principais SOs do mercado *Mobile*.

O setor empresarial se apropria bastante deste modelo de desenvolvimento, no sentido de que, com o imediatismo requirido pelo mercado, um único processo de criação, ao invés de dois (um para cada SO), facilita e agiliza tanto o gerenciamento, como o *know how* em um setor de desenvolvimento de uma empresa de software. A Tabela 3 mostra exemplos de dois *frameworks* híbridos.

Tabela 3 – Frameworks híbridos

Nome	**Mantenedor**	**Exemplos de sistemas que o utilizam**
Flutter	Google LLC	Alibaba Group, Google Pay, Nubank
React Native	Meta, Inc.	Facebook, Microsoft Office, Discord

Fonte: Elaborado pelo autor.

2.2.3 Web Apps

Com a crescente demanda pelo desenvolvimento hibrido, notou-se também o seu alto custo de produção, principalmente falando de capacitação profissional. Logo, por sua vez, a Google lançou um conceito chamado Progressive Web App (ou só Web App), este, que é feito como uma página web desenvolvida com JavaScript, HTML, e CSS, por exemplo, funcionando como um aplicativo *Mobile*. Em outras palavras, é como se o aplicativo funcionasse dentro de um navegador web do celular, e sua parte nativa fosse apenas esse envelopamento da página web, onde é nesta página que de fato ocorre o funcionamento do aplicativo. Contudo, o desenvolvimento por Web Apps é voltado para oferecer uma experiência mais semelhante possível ao da utilização de um aplicativo nativo, com esse processo ocorrendo de forma transparente, no sentido de evitar que um usuário comum note se está usando uma aplicação web app ou nativa (FORTUNATO; BERNARDINO, 2018).

2.3 App Inventor

Já listados os principais SOs e tecnologias *Mobile*, pode-se então descrever a ferramenta MIT App Inventor, e como ela se encaixa nesse contexto. O App Inventor é uma plataforma de desenvolvimento de aplicativos de celular, através de programação por blocos.

Programação por blocos é um método alternativo de se programar, funcionando por meio de gestos de clique, arrasto, e encaixe de blocos. Cada bloco tem um respectivo papel no funcionamento da aplicação, podendo ser de diversos tipos, no App Inventor existem blocos que tratam de estrutura de controle, repetição, funções, de texto, blocos para tratar expressões lógicas, navegação entre telas, dentre outros. São apresentados nas figuras a seguir (3, 4, 5, 6), dois exemplos de aplicativos criados com o App Inventor, um deles é feito para somar dois números, e o outro recebe uma cadeira de caracteres do usuário e a mostra na tela.

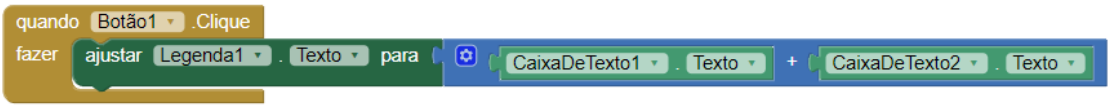
Figura 3 – Funcionamento do Aplicativo Soma Dois Números

(a) Tela inicial

(b) Após ação

Fonte: Elaborado pelo autor.

Figura 4 – Programação em blocos do Aplicativo Soma Dois Números

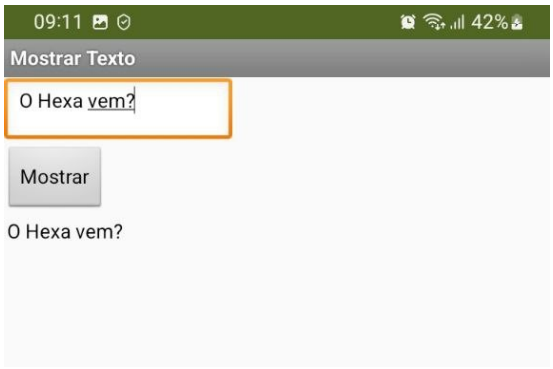


Fonte: Elaborado pelo autor.

Figura 5 – Funcionamento do Aplicativo Mostrar Texto

(a) Tela inicial

(b) Após ação



Fonte: Elaborado pelo autor.

Figura 6 – Programação em blocos do Aplicativo Mostrar Texto

Fonte: Elaborado pelo autor.

O interessante sobre o App Inventor é o fato de possuir um simplificado processo de configuração do ambiente para estudos, facilitando a quebra da barreira do início da aprendizagem de programação. O estudante pode executar os aplicativos criados rapidamente no seu próprio celular. Para conectar seu celular à plataforma e executar os aplicativos, basta o estudante conectá-lo na mesma rede Wifi que o computador, e realizar o pareamento por *QR Code*, ou pode, se preferir, usar um cabo USB para ter uma conexão mais estável com computador.

A plataforma possui capacidade de acesso e manuseio de dispositivos de mídia do celular, como câmera, gravador de som, e também a diversos outros sensores presentes no dispositivo móvel. Isto permite um desenvolvimento com amplas possibilidades, além de aumentar o entusiasmo do desenvolvedor com sua criação.

Em conjunto a isto, o fato do App Inventor não ter restrições de funcionamento nos principais SOs *Mobile*, faz com que ele se enquadre sendo uma plataforma Híbrida de funcionamento. Contudo, o processo de gerar o executável do aplicativo e o lançar na loja do respectivo SO do celular, ainda é muito mais simples no ambiente Android, do que no iOS.

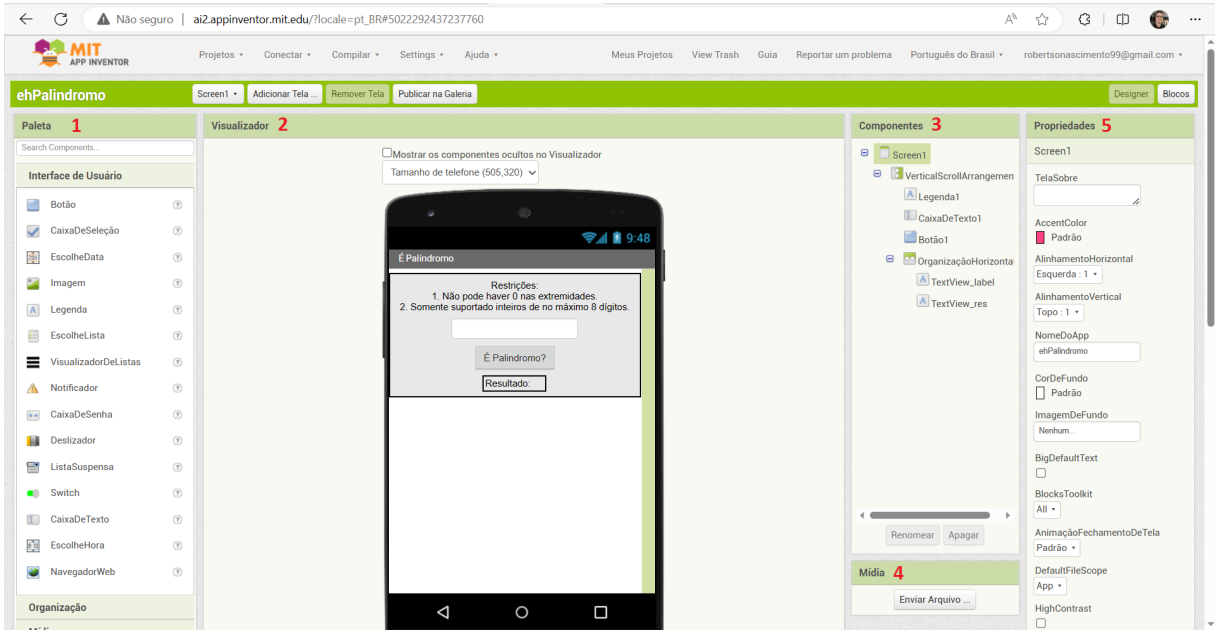
2.3.1 App Inventor em Partes

Para desenvolver um aplicativo com o App Inventor, é preciso que o programador aprenda a utilizar sua interface, esta que pode ser definida dividindo-se em dois painéis principais. Estes que são: painel de Design, e painel de Blocos.

2.3.1.1 Painel de Design

Aqui é onde ocorre o gerenciamento estático dos componentes presentes no aplicativo. Por onde se definem quais eles serão, e como serão renderizados na tela, contudo, também podem existir componentes invisíveis. O Painel de Design (Figura 7) se distribui em cinco outros painéis, sendo eles:

Figura 7 – Painel de Design do App Inventor



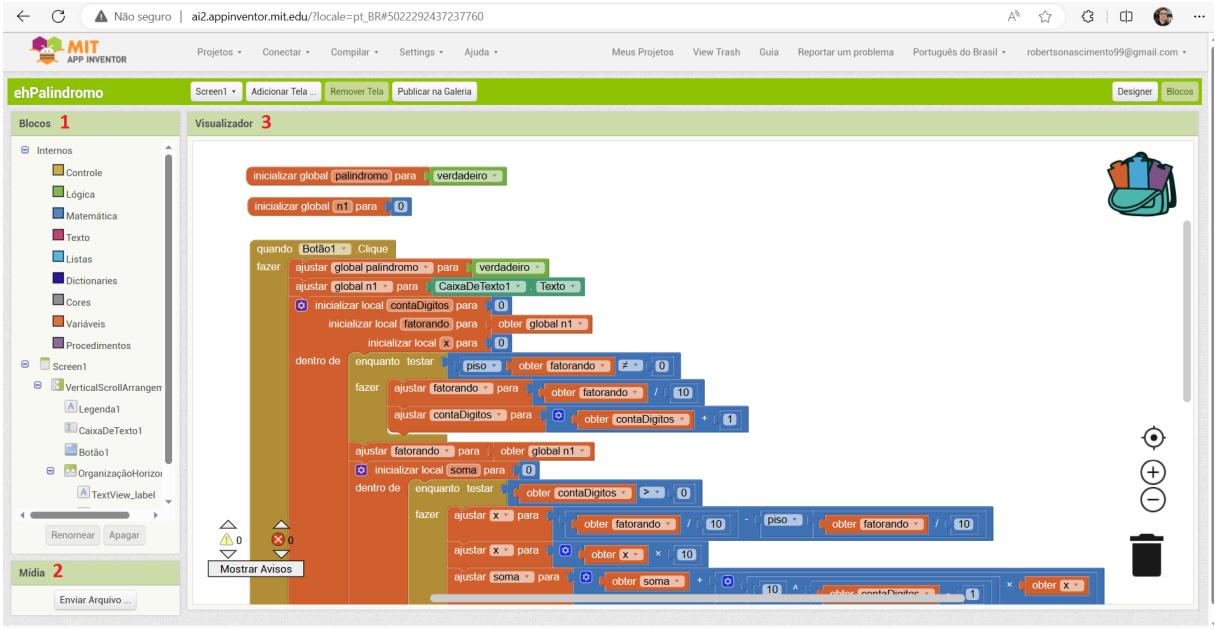
Fonte: Elaborado pelo autor.

1. Paleta: onde o desenvolvedor tem a lista de todos os componentes disponíveis para sua utilização na criação das telas e funcionalidades do aplicativo. Sendo lá também onde estão presentes os componentes de *User Interface* (UI), que permitem a interação do usuário com o sistema, e também fornecem informações metalinguísticas que servem como instruções da utilização do sistema;
2. Visualizador: onde o desenvolvedor pode ver uma prévia de como os componentes de UI estão localizados na tela do celular. Com opções de exibição de tamanho de tela em formato celular, tablet, e monitor;
3. Componentes: basicamente é a listagem dos componentes que se encontram na aplicação. O desenvolvedor pode então facilmente selecionar e gerenciar o componente que desejar.
4. Mídia: listagem dos recursos de mídia importados para o projeto do aplicativo. Por exemplo, arquivos de áudio, e imagens.
5. Propriedades: as características correspondentes ao componente selecionado no painel de Componentes. Aqui, é onde o desenvolvedor pode editar a maneira como a qual ele deseja que o respectivo componente se comporte (ainda não sendo a parte lógica).

2.3.1.2 Painel de Blocos

Parte que trata da lógica de execução do *software*. Tudo que o desenvolvedor deseja que seu aplicativo faça do ponto de vista lógico, ele precisa programar por blocos neste painel. Perceba que, os blocos possuem um comportamento específico, tendo que respeitar algumas regras de encaixe. Tais regras, são representados por cores e formas de encaixe que cada bloco possuí, passando então de maneira explícita ao desenvolvedor o modo como o respectivo bloco deve ser usado. Assim como o de Design, o Painel de Blocos (Figura 8) se divide em partes:

Figura 8 – Painel de Blocos do App Inventor



Fonte: Elaborado pelo autor.

1. Blocos: se dividindo em três tipos: blocos internos, onde estão os blocos padrões de utilização na plataforma, por exemplo, de estrutura de controle, repetição, matemáticos, e de tratamento de texto; de tela, os quais são os blocos correspondentes aos componentes contidos na aplicação; e qualquer, uma forma de programar um comportamento genérico para todos com componentes de determinados tipos.
2. Mídia: listagem dos recursos de mídia importados para o projeto do aplicativo. Por exemplo, arquivos de áudio, e imagens.
3. Visualizador: onde de fato ocorre a programação visual por blocos.

2.4 Ensino de Programação

Este, que é um dos conceitos-chave do corrente trabalho, pode ser desenvolvido de várias formas, e em períodos distintos, no que se refere a vida estudantil dos estudantes de programação. Visto que alguns indivíduos já estudam programação desde o primário, e outros apenas adentram à área quando chegam na universidade. É então importante que ao se pensar neste campo de ensino, o meio acadêmico perceba tal situação e aborde os conceitos de programação de maneira gradual e evolutiva.

Ao começar a programar algo, é primeiro necessário que o programador chegue a um algoritmo fruto do seu próprio raciocínio lógico. Seguinte a isto, ele elabora de algum modo (através de uma linguagem de programação) uma sequência de passos que devem ser seguidos para que o algoritmo como um todo funcione corretamente. Além do conceito de algoritmo, outros conceitos como o de dados, processamento, informação de saída, e tratamentos de erros estão presentes no cenário de ensino de programação.

2.4.1 Algoritmos

Representação de uma determinada sequência de passos que precisa ser seguida para que se chegue a um respectivo resultado, é o que chamamos de algoritmo. Estes passos, que na maioria das vezes recebem dados, fornecidos por usuários ou outros algoritmos, realizam determinadas ações que foram previamente programadas e então devolvem as informações pós-processadas como forma de resultado para os passos subsequentes do algoritmo, para outros algoritmos propriamente ditos, ou para um ser humano usuário do sistema em questão.

O termo algoritmo é comumente usado em Computação, porém ele pode ser pensado e ensinado do ponto de vista de várias áreas. Por exemplo, um mecânico de automóveis ao trocar a embreagem de um carro, realiza o algoritmo correspondente para tal ação, que pode ser, a grosso modo:

1. Passo 1: abrir tampa do motor;
2. Passo 2: retirar embreagem antiga;
3. Passo 3: inserir nova embreagem;
4. Passo 4: fechar tampa do motor.

A lista acima mostra então de modo simples e prático um exemplo de algoritmo.

2.4.2 Linguagens de Programação

Partindo do pre-suposto que programar é criar e fornecer instruções de processamento de dados que ficarão gravadas na forma de sequência de passos de um algoritmo. Vem a tona a maneira como a qual o programador pode transcrever seus pensamentos para que o computador compute e execute determinada tarefa. Neste contexto se enquadram as linguagens de programação.

Podendo ser de diversos tipos, dentre as linguagens mais comuns atualmente, podemos destacar: JavaScript, TypeScript, e PHP no contexto empresarial; C, C++, e Python no contexto acadêmico. Contudo, cada linguagem carrega sua respectiva característica, vantagens, e desvantagens. Por exemplo, algumas propriedades como a de tipagem de variáveis (se é forte, dinâmica, ou fraca) está bastante presente no fator de tomada de decisão entre a utilização de uma linguagem ou outra em um determinado projeto de desenvolvimento, ou ensino.

Assim como a tipagem, devemos esclarecer que existe um ponto crucial no que diz repeito ao poder de controle computacional de uma linguagem, podendo ser de alto nível, baixo nível, ou em algum lugar da escala entre estes. Resumidamente, quanto mais alto é o nível de uma linguagem, mais abstrata ela é, ou seja, a distância das instruções interpretadas pelo computador até o que foi fornecido pelo programador é maior. Um exemplo claro disto é a maneira como declarar uma variável, ler o valor do usuário e atribuí-lo a variável, em C o trecho de código é: int v1; scanf("%d", &v1). Já em Python resumi-se a: v1 = input(). Semanticamente, os dois códigos fazem o mesmo, porém a linguagem C, por possuir tipagem de nível médio, exige manipulação de endereços de memória, por exemplo, tendo assim um código mais rustico e não intuitivo para quem olha à primeira vista. Assim, o código em Python é mais legível, mas por conta de resumir algumas coisas ao contrário do C, o controle sobre algumas ações de como o código irá funcionar é menor, sendo assim um código mais genérico, que tanto pode ser bom, quanto ruim, dependendo do contexto da aplicação. Um ótimo exemplo para demonstrar este ponto é o surgimento da linguagem TypeScript, que é apenas uma forma de JavaScript com tipagem estática, e mais de baixo nível do que o JavaScript, assim ganhando em robustez e melhor tratamentos de erros, por exemplo.

No campus da Universidade Federal do Ceará (UFC), em Quixadá, geralmente o ensino de programação é iniciado em linguagens como C, e C++, onde o C possui tipagem de nível considerado médio, e o C++ sendo fortemente tipado, com algumas exceções, podendo ser em Python. Ensinar programação em C pode ser mais proveitoso em cursos de Engenharia de

Computação, por exemplo, pois os alunos devem entender e se preocupar com o gerenciamento de memória e outros aspectos mais próximos ao hardware. Principalmente no sentido de ensinar alguns conceitos como o de ponteiros, ou manipulações de estruturas de dados como listas, levando o estudante a pensar em como gerenciar de melhor forma os componentes de baixo nível que a linguagem C permite acesso. Enquanto estudantes de Design Digital podem ver disciplinas de introdução a programação em Python, por exemplo, pois sua área de estudo é mais voltada para interfaces visuais e não gerenciamento de recursos de baixo nível de programação. Desta forma, dando maior foco aos conceitos de lógica de programação, sendo este o alvo principal dos professores de disciplinas que ensinam a programar.

2.4.3 Célula de Estudos

Neste trabalho, uma célula de estudos é definida como um grupo de estudos ministrado por um estudante de graduação. Desta forma, este grupo é destinado a que estudantes também de graduação consumam os materiais ofertados pela célula, visando praticar atividades estudantis que os agreguem conhecimento.

Organizada na forma de encontro semanal, com duração de duas horas de conversação, com uma célula de estudos é possível haver uma abordagem alternativa de atividades e práticas com os estudantes de graduação sobre um determinado tema. No caso deste trabalho, o tema da célula de estudos será o ensino de programação via plataforma App Inventor. Resultando em ganhos de conhecimento aos participantes de tal grupo. Possibilitando a melhoria de habilidades impostas aos estudantes durante suas carreiras acadêmicas e, às vezes, profissionais.

3 TRABALHOS RELACIONADOS

Este Capítulo apresenta os trabalhos relacionados com o corrente projeto, assim como a descrição do que foi realizado em cada um deles.

3.1 O Uso do Software Livre App Inventor no Processo de Ensino e Aprendizagem da Lógica no Curso de Graduação em Sistemas de Informação

Neste artigo Guimarães *et al.* (2016) foi feita uma pesquisa que analisou o uso da ferramenta App Inventor no ensino de programação no curso de Sistemas de Informação, feita através de questionários aplicados aos alunos, também foi notada limitações no ensino de programação nesse contexto. Além de ter sido relatado o imediatismo por parte dos estudantes na busca de aprender os conceitos iniciais de programação, mostrando assim a importância da utilização de métodos alternativos de ensino.

O artigo cita então a utilização do software App Inventor no ensino de programação, de modo a, por ser no ambiente de dispositivos móveis, agregar mais semelhança aos aplicativos do mercado atual e da realidade dos estudantes. A escolha desta ferramenta tanto aumenta a possibilidade de uso, quanto quebra barreiras no sentido de início de interação entre aluno e software, no seu processo de criação. Outro fator que resultou na escolha do App Inventor é a programação visual (ou programação por blocos) que diminui a carga cognitiva que se é preciso para desenvolver as primeiras aplicações do aluno.

A programação por blocos dá *feedbacks* de erros (quando existentes) de maneira a amenizar também a então frustração ressaltada por Wolber *et al.* (2011), que um desenvolvedor tem ao não conseguir identificar erros lógicos na programação.

Primeiramente foi apresentado os dados do projeto aos docentes do curso de Sistema de Informação, esse que haveria a criação de um aplicativo móvel usando o App Inventor. Ao final os trabalhos foram apresentados com sucesso em uma feira. Como forma de avaliação deste projeto, foi feita a aplicação de um questionário com os estudantes que fizeram parte do mesmo.

O questionário foi aplicado a 15 alunos voluntários e continha perguntas como: 1) Você já possuía algum conhecimento anterior no uso do software?; 2) Você considera fácil utilizar o software para se criar um aplicativo para celular? 3) O software favorece a memorização dos recursos de programação que podem ser utilizados?

Os resultados da pesquisa mostraram que a maioria dos alunos (83%) concordou

que o App Inventor contribuiu efetivamente para a aprendizagem de lógica de programação do aplicativo desenvolvido pelo projeto. Entretanto, a maioria também não ficou satisfeita com o resultado final da aplicação, considera-se que isso é fruto do aplicativo não estar completamente concluído, passando ainda por melhorias em suas qualidades frente a outros aplicativos semelhantes.

Concluiu-se que a utilização do App Inventor contribuiu positivamente para o aprendizado de lógica de programação. Levando em consideração o novo ambiente de estudos dos alunos, que é o universitário, onde é exigido maior capacidade de concentração, abstração, dentre outras, o uso do software foi satisfatório.

Diminuindo a memorização de comandos, a ferramenta possibilitou um maior foco por parte dos alunos na lógica de programação. Também o App Inventor se mostrou vantajoso em relação aos seus concorrentes por não necessitar de instalação de programas complementares (Plugins).

Ao contrário do estudo de programação do ponto de vista apenas teórico, se mostra positivo um caminho para ensino de lógica, baseado em softwares mais amigáveis, e com foco em resultados práticos.

3.2 O Uso da Plataforma MIT App Inventor no Incentivo a Inserção de Alunos do Ensino Médio ao Ensino Superior

Neste trabalho Vieira *et al.* (2020) foi realizado um minicurso com alunos do ensino médio com o intuito de lhes incentivar a ingressar em uma universidade. O minicurso se mostrou possibilitar além de uma boa interação entre os universitários com os alunos de ensino médio, também de um modo geral a melhoria da didática dos discentes e maior interesse dos alunos pelo mundo acadêmico.

Considerando o que foi afirmado pelo autor:

> Marcelo Knobel (Pró-Reitor de Graduação da UNICAMP no ano de 2009 a 2013) durante o simpósio Excellence in Higher Education no ano de 2014 afirmou que “A grande massa de estudantes que concluem o ensino médio em escolas públicas não considera o ingresso em universidades públicas, pois sabe que tem pouca ou nenhuma chance de entrar nessas instituições". (VIEIRA *et al.*, 2020, p. 76524).

Desta forma, fica clara a necessidade da existência de projetos de extensão que aproximem o ensino superior da realidade dos alunos do ensino médio, seja por meio de palestras ou minicursos.

O trabalho de Vieira *et al.* (2020) concluiu também que a maneira como a qual o minicurso foi ministrado, teve além de ensinar a capacidade técnica, ressaltar a importância de que "para além da formação numa profissão específica, deve ser considerada a formação de cidadãos conscientes de sua responsabilidade social" (BENETTI *et al.*, 2015). Tendo assim um resultado positivo no lado social e de metodologias de didática aos discentes ministrantes do minicurso.

Foram conceituados a ferramenta App Inventor, e programação por blocos. Evidenciando o potencial educacional possibilitado pela ferramenta, do ponto de vista de funcionalidades interativas do aparelho celular, como comando de voz e GPS. Além disso, foi definido seu ambiente gráfico como atraente e motivador no processo de aprendizagem dos conceitos de lógica de programação no ensino médio e superior (GOMES; MELO, 2013).

A priori foi feito o planejamento de como o minicurso seria aplicado: qual escola de ensino médio seria contemplada com o projeto, duração, e dias da semana. Tendo então sido definido um total de duas aulas, em dias diferentes, com a duração de 2 horas e 30 minutos para cada turma. Usando a documentação do próprio site do App Inventor, assim como de outros relacionados ao seu conteúdo, foi feita a capacitação dos autores (discentes universitários) do projeto.

Foi proposto então, para os responsáveis das possíveis escolas contempladas, o projeto de extensão Elaboração de aplicativos para Android com o uso do App Inventor: Uma Experiência em Busca da Segurança nos Ônibus, obtendo informações sobre os detalhes (laboratório de informática e horários disponíveis) da possibilidade da realização do minicurso.

Com a aprovação da escola, foi possível a apresentação do projeto para os alunos, que se, caso se interessassem, poderiam se inscrever no projeto durante o turno oposto ao de suas aulas normais. O aplicativo escolhido para ser desenvolvido no projeto foi uma Agenda de compromissos. As aulas foram ministradas por discentes de diferentes graduações da Universidade Federal do Pará do Campus Tucuruí. Os alunos concluíram o curso com presença mínima de 75% e receberam certificado de participação.

O minicurso foi realizado no mês de junho de 2018, na escola de ensino médio EEEM Rui Barbosa, localizada na cidade de Tucuruí - PA, e possuiu um total de 14 alunos inscritos, todos do terceiro ano, dos quais 12 deles receberam o certificado de conclusão.

No decorrer das aulas, foi percebido um grande interesse, por parte dos alunos, no desenvolvimento de seus próprios aplicativos. Assim como, surgimento de dúvidas sobre

quais tipos de aplicativos poderiam ser criados utilizando o App Inventor, e a possibilidade de aplicativos ajudarem nos seus cotidianos. Desta forma, o conhecimento adquirido no minicurso resultou no aumento do interesse dos alunos por programação e criação de outras aplicações.

Pela quantidade de participantes ter sido relativamente baixa, os processos de acompanhamento, por parte dos ministrantes, foram possíveis serem realizados mais de perto, retirando dúvidas e facilitando com erros ou problemas na etapa de programação. A ajuda mútua se mostrou sempre presente entre os alunos, destacando o fator social e exercício da cidadania proposto pelo projeto. O minicurso, tendo sido ofertado em duas turmas, possibilitou um aprendizado empírico aos ministrantes, de modo a haver uma melhoria contínua na qualidade das aulas.

A liberdade de criação de aplicativos trazida pelo App Inventor, no sentido de *Layout* gráfico, demonstrou despertar interesse nos alunos pela área de computação. Os exercícios propostos foram aceitos e concluídos pelos estudantes, sem muitas dificuldades, evidenciando a facilidade na utilização da ferramenta, mesmo os alunos que não tinham contato prévio com programação. Após o término da criação do aplicativo, houve a fase de testes, tendo sido realizada nos próprios celulares dos alunos, onde os resultados foram considerados satisfatórios.

Os resultados obtidos demonstram a importância de projetos de extensão universitária aproximarem os alunos de escolas públicas do ensino superior. A interação entre discentes, de diferentes cursos e áreas, com estudantes de ensino médio, se mostrou positiva, desmistificando a ideia de que o ensino superior é demasiado difícil e distante da realidade. Para os bolsistas e voluntários ministrantes do projeto, a melhoria da prática didática e do trabalho em equipe foram notadas, além do agregamento do conhecimento de desenvolvimento de aplicativos com o App Inventor.

3.3 Why is programming so difficult to learn? Patterns of Difficulties Related to Programming Learning

Este artigo Bosse e Gerosa (2017) é um dos frutos da tese de doutorado da Professora de Ciência da Computação, da Universidade Federal de Mato Grosso do Sul. Ele organiza e relaciona padrões de dificuldades presentes no contexto de dificuldade no aprendizado de programação, com a finalidade de facilitar no processo de foco dos alunos, e também auxiliar professores na sua metodologia de ensino, e informar pesquisadores sobre o assunto. As questões de pesquisa deste artigo foram:

- QP1 Qual é a taxa de insucesso nos cursos de introdução à programação?

- QP2 Quais são as dificuldades relatadas na literatura para aprender a programar?
- QP3 Quais são as dificuldades de aprender a programar na perspectiva dos alunos?
- QP4 Quais são as dificuldades de aprender a programar a partir da perspectiva dos instrutores?
- QP5 Que erros de sintaxe e semântica são recorrentemente encontrados no código desenvolvido pelos alunos?
- QP6 Como aplicar os padrões identificados para melhorar o ensino-aprendizagem de programação?

Antes de mostrar alguns resultados na obtenção das respostas das questões de pesquisa, o artigo mostra a metodologia que foi seguida. Informando que, como o artigo faz parte de uma pesquisa maior, os autores se referem a obtenção de alguns resultados no futuro. Portanto, não apresentando respostas para todas as questões de pesquisa.

Para responder a QP1 foi feita uma consulta no banco de dados da Universidade de São Paulo (USP) pelas disciplinas de Introdução a Programação entre os anos de 2010 a 2014, sendo feita a consulta individual dos resultados de cada disciplina. Os resultados corroboraram com os estudos referenciados pelo artigo. Dos 18.784 cadastros realizados no período analisado, 30% resultaram em reprovações ou desistências, o que se manteve bastante constante ao longo dos anos. Foi evidenciada uma taxa de reprovação maior para os alunos que não eram da área de computação, chegando a 30,3% contra 25,1% dos alunos que são da área. Também foi descoberto que mais de 25% dos aprovados frequentaram duas vezes essas disciplinas. Logo, foram considerados altos os índices de insucesso (reprovação ou desistências), em relação a disciplinas de outras áreas de nível semelhante.

Para responder a QP3 foi aplicado uma técnica de diários de estudos a 34 estudantes, no segundo semestre do 2015, de seis diferentes cursos, onde foi percebido a predominância de erros de sintaxe, em comentários de estudantes do tipo *continuo tendo erros de sintaxe, o código não executa e eu simplesmente não sei como resolver*. Estes, que atrasam a análise da parte lógica do código. Outras dificuldades foram apresentadas relacionadas a variáveis, com comentários do tipo *não entendi a diferença de tipo int para float, tive que testar em um programa para notar claramente*. Assim como dificuldades relacionadas ao ambiente de desenvolvimento, e a falta de auxílio nas mensagens de erro fornecidas pelas IDEs.

Para fins de uma análise aprofundada foi utilizada também outra abordagem de observação dos alunos, onde foram realizadas seis entrevistas com seis alunos que participaram da prova final da disciplina de Introdução a Programação. Nesta entrevista foram aplicados

testes de lógica de programação aos alunos, com dificuldade crescente, onde foram gravados os resultados, a tela dos computadores dos alunos, e suas reações. Uma observação percebida foi que dois dos seis estudantes, anotaram ideias de como resolver o problema enquanto liam a definição do problema. Eles não obtiveram necessariamente notas melhores ou piores do que de outros estudantes, porém, tiveram menos erros com declaração de variáveis, por exemplo. Sinais de criação de barreiras mentais de conhecimento foram notados, onde dos seis alunos, alguns fizeram comentários como *fico nervoso ao ver a palavra matriz*, isto se mostrou estar relacionado ao sucesso em resolver o problema, pois estes alunos que comentaram negativamente sobre o anunciado de uma questão não a conseguiram resolver.

Foi notado inclusive que ao se depararem com erros de semântica, os estudantes se desestimulavam mais facilmente, pois iriam levar mais tempo para os resolver.

Respondendo a QP4, para entender a perspectiva dos professores no que concerne a dificuldade dos alunos em aprendizagem de programação, foram feitas entrevistas individuais com 14 professores do departamento de Ciência da Computação da USP. A análise dessas entrevistas foi realizada, conforme os autores, por meio do procedimento Grounded Theory. Nas entrevistas, alguns professores relataram o uso de pseudocódigo, outros informaram que preferem partir direito para a linguagem de programação. O principal problema citado foi o de dificuldade com o *pensamento lógico*, e também problemas na identificação da precedência das operações matemáticas, e valores lógicos da tabela verdade.

Sobre os conceitos fundamentais de programação, alguns professores citaram a preferência pela utilização do comando *while*, no lugar do *for*, pois o mesmo foi notado de melhor entendimento pelos alunos. Também um conteúdo dificultoso foi considerado a mistura de repetição com estrutura de decisão. Quando se trata de vetores, foi ressaltado pelos professores que é nesta matéria que os alunos precisam demasiadamente praticar.

Sobre funções, os professores acreditam que a maior dificuldade está no escopo das variáveis, e importância do valor de retorno. Parâmetros não apresentaram gerar problemas, porém, o grau de dificuldade aumentou ao usar o conceito de passar por referência na linguagem C. Os fatores humanos ressaltados como agravantes são heterogeneidade dos grupos e entre grupos de alunos, baixa participação nas aulas, baixa frequência, turmas muito grandes, desinteresse em aprender, linguagem de programação adotada, trauma de alunos que repetem o curso, entre outras.

Ressaltado também a necessidade de geração de estímulo nos estudantes, com

a utilização de jogos e ferramentas não tradicionais de programação, além, do fomento ao aprendizado de como resolver problemas, e não a sua memorização. O artigo conclui que o processo feito para responder às questões de pesquisa já obteve ganhos no ensino de programação, pois foram notadas nas entrevistas que os professores falaram sobre métodos de ensino para contornar algumas das dificuldades listadas pela pesquisa.

3.4 Estudo do Uso da Linguagem de Blocos Scratch no Ensino do Pensamento Computacional

Este trabalho Rodrigues (2019) foi uma monografia de conclusão de curso que possuiu tema e metodologias bem semelhantes ao do corrente projeto, porém, se tratando da utilização da ferramenta de programação por blocos Scratch. Logo na introdução o autor ressalta a importância da capacidade de abstração, para o desenvolvimento do pensamento computacional, que, neste contexto, é equivalente à habilidade de programar. O autor também cita Brackmann (2017) onde pela trecho: "seria como se eles conseguissem ler, mas não conseguissem escrever com as novas tecnologias", se pode observar o contexto da sociedade atual, onde a maioria das pessoas tem o contato com Computação, mas pode apenas consumir aquilo, e não criar.

Continuando sobre a importância do aprendizado de programação, o trabalho adentra nas limitações das linguagens de programação comuns, em que seu entendimento não é trivial. Chegando então a programação por blocos, que de certa forma descomplica este processo de utilização de uma linguagem de programação, e proporciona maior engajamento dos estudantes na parte da construção lógica em resolver um problema computacional. O trabalho teve como principal foco responder à questão de pesquisa:

- Os participantes dos grupos de aprendizagem apresentaram bons desempenhos nas atividades envolvendo conceitos do pensamento computacional utilizando a linguagem Scratch?

Tendo então como objetivo a promoção do ensino-aprendizado do pensamento computacional nos participantes do grupo de estudos por meio da programação por blocos da plataforma Scratch. Para tal, o autor elaborou um material educativo contendo seis módulos com videoaulas e questionários que abordaram o desenvolvimento de jogos computacionais na referida plataforma.

O estudo foi aplicado sobre cinco diferentes grupos de estudos, todos eles sendo de alunos da Universidade Tecnológica Federal do Paraná do campus Campo Mourão. Cada grupo foi formado por um tipo de participante, sendo eles professores de física na disciplina de mestrado Atividades Computacionais para o Ensino Médio e Fundamental; estudantes do

1º ano do Ensino Médio com Técnico Integrado em Informática da UTFPR-CM, na disciplina de algoritmos; estudantes do 1º período dos cursos de Engenharia de Alimentos e Engenharia Química na disciplina de programação; estudantes do 1º período do curso de Bacharelado em Ciência da Computação na disciplina de algoritmos; e estudantes do 1º período do curso de Engenharia Eletrônica na disciplina de algoritmos.

Contudo, a análise dos dados somente foi sobre o grupo composto pelos estudantes do 1º ano do Ensino Médio com Técnico Integrado em Informática da UTFPR-CM, quando que, nos demais grupos, muitos participantes não realizaram as atividades, resultando em dados insuficientes. Os resultados gerais apresentaram-se satisfatórios tanto no quesito de melhoria nas habilidades de pensamento computacional, quanto no mantimento dos participantes em constante motivação no processo de aprendizagem dos conceitos de programação e Engenharia de Software, informações estas sendo confirmadas pela análise dos questionários, e dados motivacionais.

3.5 Comparação com trabalhos relacionados

Essa seção resume, por meio da Tabela 4, uma comparação entre os trabalhos relacionados com o trabalho proposto e suas diferenças envolvendo domínio, método de pesquisa e métricas avaliadas.

Além disto, descrevendo as relações entre estes trabalhos, podem-se destacar, sobre os trabalhos que fazem o uso de ferramentas de desenvolvimento de programação por blocos - Vieira *et al.* (2020) e Guimarães *et al.* (2016) com o App Inventor, e Rodrigues (2019) com Scratch, - a presença de benefícios no ensino-aprendizagem do pensamento computacional e na habilidade de programação. Já o trabalho Bosse e Gerosa (2017), que busca listar padrões de dificuldades encontradas no processo de aprender programação, está mais focado em estudar a capacidade de aprendizagem de programação como um todo, e quais gatilhos que, por partes dos estudantes, geram uma demasiada dificuldade de aprendizado nas fases iniciais das suas jornadas como desenvolvedores de software.

Os critérios de comparação da Tabela 4 são detalhados da seguinte forma:

1. App Inventor: Se o trabalho fez uso da plataforma App Inventor.
2. Participantes da pesquisa: Quem são os indivíduos que foram submetidos a pesquisa realizado pelo trabalho.
3. Produzido durante: O que foi feito durante o trabalho que possibilitou a realização da pesquisa.

4. Resultados: O que de fato pôde ser concluído após a realização do trabalho.

Tabela 4 – Comparação deste trabalho com os trabalhos relacionados

Características abrangentes	(GUIMARÃES *et al.*, 2016)	(VIEIRA *et al.*, 2020)	(BOSSE; GEROSA, 2017)	(RODRIGUES, 2019)	Trabalho Proposto
App Inventor	Sim	Sim	Não	Não	Sim
Participantes da Pesquisa	Estudantes do curso de Sistemas de Informação	Estudantes de Ensino Médio	Calouros, e Professores de cursos da área de Computação	Estudantes de nível médio, superior, e docentes de ensino superior	Estudantes de quaisquer cursos do campus UFC Quixadá
Produzido Durante	Criação de um aplicativo	Criação de um aplicativo	Pesquisa quantitativa e qualitativa	Guia de estudos de programação Scratch, videoaulas, e análise de resultados	Criação de mais de um aplicativo, Comparação de testes de lógica de programação
Resultados	Evidenciar melhoria no aprendizado de Lógica de Programação	Aumento da taxa de ingresso de alunos de Ensino Médio de escolas públicas a Universidade	Listagem de dificuldades em aprendizagem de programação	Conclusões positivas sobre a motivação dos participantes e do critério de aprendizagem	Conhecimento sobre o resultado da utilização do App Inventor sobre estudantes de Computação

Fonte: Elaborado pelo autor.

4 METODOLOGIA

Este trabalho consiste em estudar e aplicar a ferramenta App Inventor, com foco em desenvolver um método que a utilize como forma de ensino de programação para alunos de disciplinas de introdução a programação no campos UFC Quixadá. Sendo assim, serão criados módulos de conteúdos das disciplinas de introdução a computação dentro do App Inventor, esses que serão: Variáveis; Funções; Estruturas de controle; e Repetição. As principais etapas deste projeto estão listadas cronologicamente a seguir:

1. Realização de célula de estudos: Aprendendo a Programação com App Inventor, desenvolva seus próprios aplicativos de celular;
2. Disponibilização do material de estudos on-line;
3. Aplicação de testes de conhecimentos com os alunos de dois grupos (participantes, e não participantes da célula);
4. Comparação dos resultados dos testes.

Nas próximas Seções desde capítulo, será detalhado o que será feito em cada uma destas etapas acima.

4.1 Célula de Estudos

A célula abordará os assuntos vistos nas disciplinas de introdução a programação, porém, do ponto de vista de criação de aplicativos com o App Inventor. A maneira como serão abordados os conteúdos foi separada nos módulos de estudos listados nas subseções a seguir.

4.1.1 Módulo 1: Variáveis

Este que é um conceito base se tratando de programação. É importante que seja apresentado logo no início da célula, pois tanto é, comparado aos demais, um conceito de fácil entendimento, como é essencial para no aprendizado de programação e criação de aplicativos.

Contudo, no mundo da programação, o conceito de variáveis se estende se tratando de tipografia. Mesmos após explicados os principais tipos de variáveis, sendo eles: int, float, double, char, string, e boolean, se torna necessário apresentar as variações de seus modificadores de acesso, por exemplo.

Boas práticas de programação também serão citadas, como a correta organização dos blocos na tela (que seria equivalente à indentação correta em linguagem codificada), e uma

clara nomenclatura de variáveis. Conceitos como o de eventos, e também design de telas serão explicados de maneira resumida para possibilitar, além da parte lógica de programação por blocos, um desenvolvimento de Apps de modo organizado e funcional.

Nesta etapa será criado com os alunos na célula um total de um aplicativo que envolve o conceito de variáveis.

4.1.2 Módulo 2: Funções

Este que será o segundo conceito abordado é, certas vezes, subestimado pelos professores nas etapas iniciais no ensino de programação. Para o aluno, entender corretamente como e quando utilizar funções é crucial para a construção de softwares de quaisquer tipos.

O entendimento correto deste conceito melhora a capacidade do aluno no pensamento de desenvolvimento, além disso, o faz pensar sobre organização de responsabilidades funcionais no software. Também desperta o interesse por reaproveitamento de código.

Os aplicativos criados neste módulo serão dois, onde um deles conterá o conceito de desenvolvimento *Mobile* de navegação entre telas.

4.1.3 Módulo 3: Estruturas de Controle e Repetição

Aqui então já ocorre a maior presença de lógica de programação, onde serão modelados aplicativos estritamente relacionados a problemas de lógica de programação. Em outras palavras, aplicações que se propõem a resolver exercícios de disciplinas de introdução a programação.

Serão criados três aplicativos, voltados principalmente para a prática dos estudantes.

4.1.4 Ao Final da Célula

Contudo, de forma adicional à célula, será apresentado o código fonte de um aplicativo semelhante aos já citados, porém, criado por programação codificada (programação comum) em React Native - escolhido por motivo do autor deste trabalho possuir experiência com tal -, que é um *framework* com presença crescente no mercado, e de desenvolvimento híbrido. Servindo assim como uma amostragem para os estudantes, a realização de uma comparação de como é desenvolvido um aplicativo por meio de programação por blocos - App Inventor, e por programação codificada - React Native. As figuras a seguir (9, 10, 11) ilustram exemplos de duas

aplicações, uma criada na linguagem C, e outra criada com o App Inventor, estas que realizam a mesma função de receber uma sequência de dois números inteiros, e os mostrar tela de forma ordenada.

Figura 9 – Código em C que ordena dois números inteiros

OrdenaEmC.c

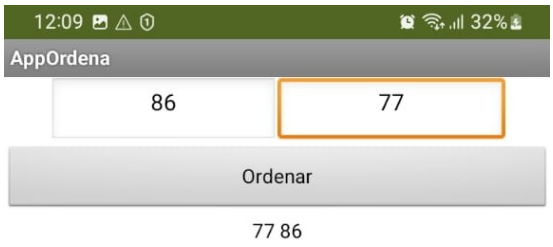
```c
#include <stdio.h>

int main(){
  int n1 = 0, n2 = 0;
  scanf("%i %i", &n1, &n2);

  if(n1 <= n2){
    printf("%i %i", n1, n2);
  }
  else {
    printf("%i %i", n2, n1);
  }
  return 0;
}
```

Fonte: Elaborado pelo autor.

Figura 10 – Funcionamento do Aplicativo Ordena Dois Números

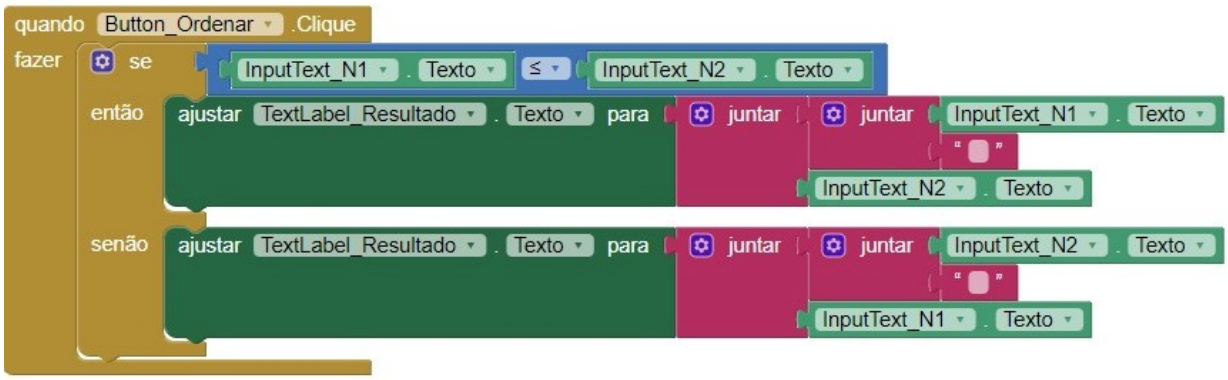
(a) Tela inicial

(b) Após ação



Fonte: Elaborado pelo autor.

Figura 11 – Programação em blocos do Aplicativo Ordena Dois Números



Fonte: Elaborado pelo autor.

Pode-se então perceber a diferença entre um algoritmo feito em C, em relação a um feito em programação por blocos. Em C o estudante já começa precisando abstrair alguns conceitos, como o de *include* e de leitura e escrita utilizando parâmetros específicos para as funções *scanf* e *printf*. Já na programação por blocos, ocorre sim uma abstração, porém de maneira mais lúdica na forma de entendimento do que o algoritmo está fazendo. Na programação em blocos da Figura 11 os componentes estão com suas identificações renomeadas, a fim de facilitar sua identificação na estruturação lógica de execução.

4.2 Disponibilização do material de estudos

O material será postado e mantido em um repositório de propriedade do autor deste trabalho, no site GitHub.

A forma de divisão do material será feita semelhante aos módulos da seção anterior. Cada módulo conterá a sua motivação de aprendizado, conceituação teórica, e então, vídeos (feitos pelo autor) da criação de aplicativos exemplos, que fundamentalmente utilizam os conceitos do respectivo módulo. Estes aplicativos serão feitos usando o App Inventor.

Serão propostos exercícios baseados nesses módulos, que consistem no estudante criar aplicativos com o App Inventor que realizam determinada tarefa, onde, essa tarefa é projetada para exigir a utilização do conceito de programação visto no módulo do exercício.

4.3 Aplicação de testes de conhecimentos com os alunos de dois grupos

Esta etapa tem por objetivo coletar as respostas de testes de motivação e conhecimento de programação dos estudantes. Estes testes serão compostos essencialmente de perguntas mais relacionadas as impressões pessoas dos alunos sobre programação por blocos e codificada, e lógica de programação, com questões de maioria subjetivas. Os testes serão aplicados de maneira remota, com dois grupos de alunos, um grupo participante da célula e consumidor do material online, e o outro não.

Detalhando mais estes testes, eles serão aplicados aos alunos no decorrer do semestre, tendo como foco mostrar sua motivação e engajamento de aprendizado dos conteúdos de programação. Totalizando dois pequenos testes (Apêndice A), onde o primeiro será aplicado na metade do semestre, e o segundo mais ao final do semestre. Com o primeiro sendo mais voltado ao engajamento sobre a área de computação, tratando se o App Inventor surtiu sobre os alunos um efeito positivo no interesse pelo aprendizado de programação. E o segundo sobre alguns conceitos de fundamentos de programação propriamente ditos.

4.4 Comparação de resultados dos testes

Esta comparação tem a finalidade de mostrar o efeito da utilização da ferramenta App Inventor como método de ensino de programação sobre os alunos participantes da célula de estudos. Comparação esta que será feita via análise de dois critérios, sendo o primeiro a quantidade de acertos e erros de cada questão dos respectivos testes nos dois grupos, e o segundo pela análise geral da diferença e semelhança entre as respostas discursivas dos dois grupos, também para cada questão dos respectivos testes.

É importante ressaltar que este não é um trabalho que segue estritamente o método científico. Para de fato possuir método científico, este experimento deveria ser realizado de forma mais controlada, formal, e possuir melhores aspectos que facilitassem sua reprodutibilidade.

5 RESULTADOS

Ao decorrer do semestre que se passou este projeto (2023.1), logo no início, houve uma divulgação do acontecimento da célula de estudos - Aprendendo a Programação com App Inventor, desenvolva seus próprios aplicativos de celular - nas redes sociais do campus UFC Quixadá, tanto no Instagram, quanto no Facebook do campus. O autor também divulgou em suas redes sociais, assim como chegou a passar em salas de disciplinas de introdução a programação para convidar os alunos pessoalmente.

No primeiro dia marcado para o início da célula, logo após a divulgação, não compareceu nenhum aluno, nas duas semanas seguintes também não. Talvez possamos citar que isto seja fruto da divulgação ter sido feita majoritariamente de forma online, e do autor ter ido pessoalmente a apenas três salas de aula da disciplina de Fundamentos de Programação (FUP). Mas o que importa é que não existiu engajamento inicial por partes dos estudantes, para o início da célula. Contudo, ao procurar alternativas de como proceder o projeto, o seu autor juntamente com seu orientador, optaram por realizar o experimento com participação de estudantes conhecidos do próprio autor. Logo, o autor convidou um total de cinco colegas de faculdade para participarem da célula, todos de cursos do campus UFC Quixadá, porém de cursos e períodos destintos, variando desde alunos de primeiro semestre até um aluno recém-formado. Eles então se mostraram interessados em ingressar na atividade, e curiosos sobre o uso do App Inventor.

Os encontros semanais do grupo de estudos foram então organizados de maneira remota, até por questões de conveniência para os cinco alunos poderem participar. Para fins de organização, o autor criou um formulário de inscrição na célula e o disponibilizou juntamente com um cartaz (Figura 12) posto na parede do mural de informações estudantis no campus UFC Quixadá. Para que se algum aluno extra quisesse participar da célula, bastaria ele entrar no link do Google Forms que está no cartaz e efetuar sua inscrição. Ao confirmar a inscrição, o aluno recebia o link de acesso ao grupo de Telegram da célula, onde lá ficou disponível tanto o link da playlist de gravações dos encontros[1], como o link do material auxiliar de estudos[2].

[1] Link da playlist de videoaulas dos encontros da célula: https://www.youtube.com/playlist?list=PLwBWD6dTMHUcKqfRyQ295B_mVfJxov7El

[2] Link do material auxiliar de estudos para os participantes da célula: https://github.com/robertsonasc/AprendendoProgramacaoComAppInventor

Figura 12 – Cartaz de divulgação da célula de estudos

Fonte: Elaborado pelo autor.

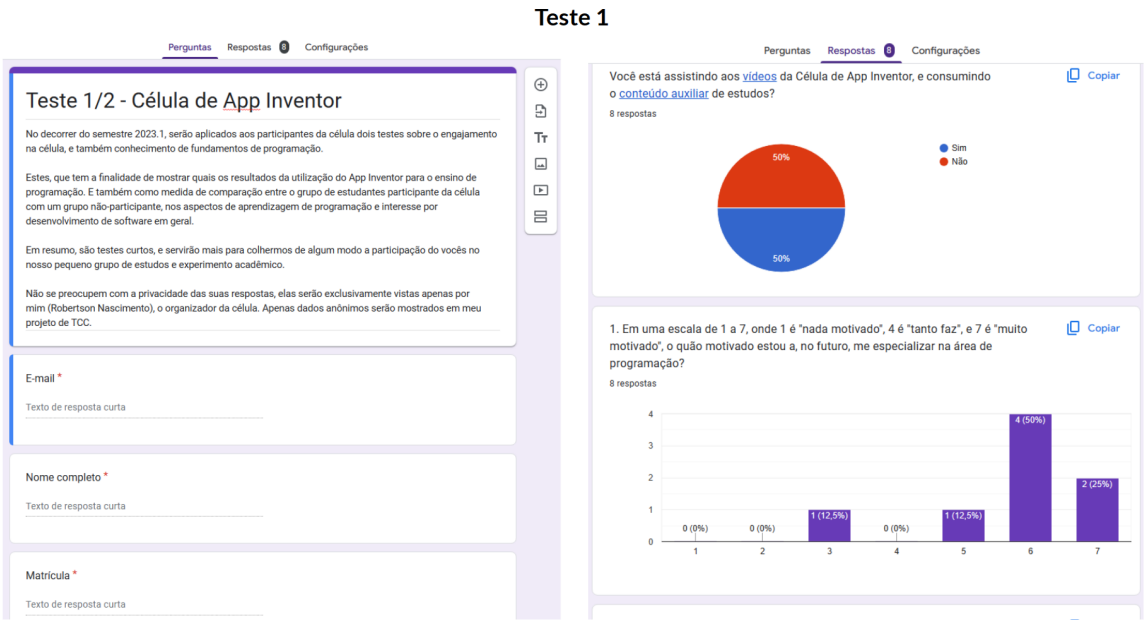
5.1 Aplicação e análise dos testes

Os testes (Apêndice A) foram voltados a colher as impressões iniciais dos alunos sobre a área de programação (codificada e por blocos), seus interesses profissionais futuros, além de testar seus conhecimentos de lógica de programação e pensamento computacional. Foram aplicados a 8 pessoas, com 4 delas sendo participantes da célula, e os outros 4 não participantes, incluindo 1 recém-formado em Engenharia de Software do campus UFC Quixadá e 7 outros estudantes também do mesmo campus de variados cursos. Pela quantidade de participantes da pesquisa ter sido considerada baixa, os resultados para a análise de impacto de tal pesquisa pôde ser classificada de nível relativamente pequeno, do ponto de vista de amostragem de dados. O método de comparação do grupo participante da célula com o não participante, consistiu em comparar as respostas de cada pergunta do teste de maneira separada.

5.1.1 Teste 1

A Figura 13 mostra uma prévia do Teste 1 na plataforma Google Forms. Já as Figuras 14 e 15 mostram de forma geral (não separado por grupos) as respostas deste teste que puderam ser organizadas em forma de gráficos. O primeiro teste conteve 4 respostas do grupo participante da célula, e 4 respostas do grupo não participante da célula.

Figura 13 – Prévia do Teste 1 na plataforma Google Forms

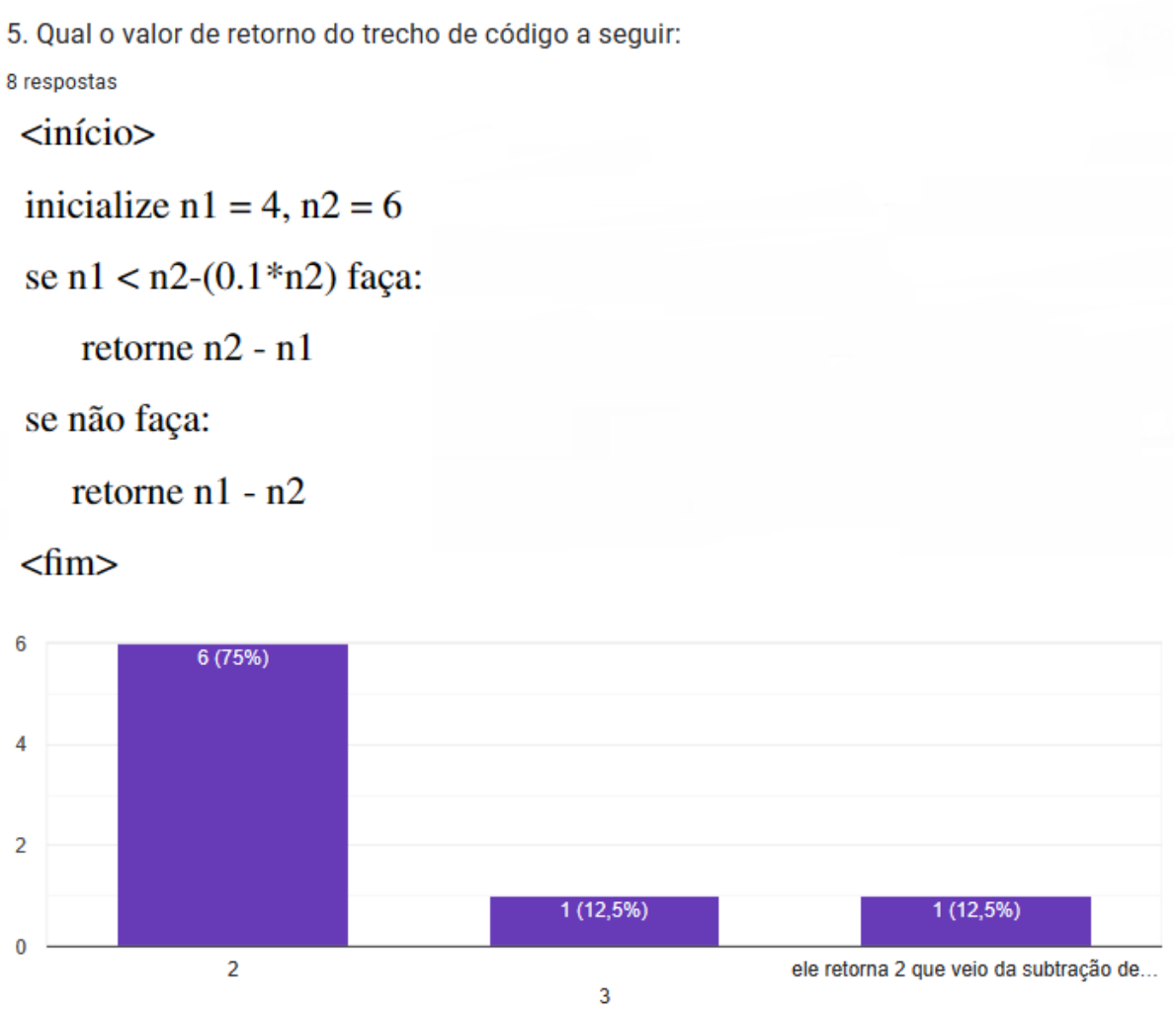


Fonte: Elaborado pelo autor.

Figura 14 – Gráfico 1 de respostas

Fonte: Elaborado pelo autor.

Figura 15 – Gráfico 2 de respostas



Fonte: Elaborado pelo autor.

Conforme as respostas das 7 perguntas que o compuseram, os resultados puderam ser categorizados da seguinte forma:

Questão 1 - Quantos estudantes se interessam por se especializar na área de programação no futuro, em uma escala de 1 a 7, onde 1 é nada motivado, e 7 é muito motivado? (Tabela 5).

Tabela 5 – Análise da Questão 1 - Teste 1

Grupo participante da célula	
Nº de respostas	**Respostas**
2	7
2	6
Grupo não participante da célula	
Nº de respostas	**Respostas**
2	6
1	5
1	3
Análise: Por parte do grupo participante da célula, foi notada um interesse maior por ingressar na área de programação no futuro.	

Fonte: Elaborado pelo autor.

Questão 2 - Quantos alunos tiveram contato com programação antes de entrarem na universidade? (Tabela 6)

Tabela 6 – Análise da Questão 2 - Teste 1

Grupo participante da célula	
Nº de respostas	**Respostas**
3	Não
1	Sim
Grupo não participante da célula	
Nº de respostas	**Respostas**
2	Não
1	Sim
1	Muito pouco, quase nada
Análise: Não se apresentou ter diferença notável sobre esta questão, visto que é comum, jovens que ingressam em graduações de computação pesquisarem um pouco sobre programação, ou até mesmo programarem problemas simples, antes iniciarem suas graduações.	

Fonte: Elaborado pelo autor.

Questão 3 - Qual sub área de estudo de Computação que mais lhes interessou até o momento? (Tabela 7). Lembrando que esta questão possuiu respostas semelhantes, então foram consideradas tais como do tipo não exclusivas[3].

[3] Respostas não exclusivas: Digamos se um aluno respondeu, que gosta de Web e Mobile, foi somado 1 tanto na resposta Web, quanto Mobile.

Tabela 7 – Análise da Questão 3 - Teste 1

Grupo participante da célula	
Nº de respostas - não exclusivas	**Respostas**
2	Web
1	Mobile
1	Banco de dados
1	Programação em geral
Grupo não participante da célula	
Nº de respostas - não exclusivas	**Respostas**
1	Programação
1	Web
1	Sem opinião formada
1	Segurança da informação
Análise: As respostas refletiram muito de cada aluno, como eles são amigos e colegas do autor deste projeto, eles possuem interesses por áreas de estudos parecidas, sendo relacionadas a programação e desenvolvimento de software em geral.	

Fonte: Elaborado pelo autor.

Questão 4 - Já haviam ouvido falar sobre programação por blocos? Se sim, explicassem o que eles sabiam que poderia ser criado com isto, se não, explicar o que eles imaginavam que era possível ser criado com tal (Tabela 8). As respostas desta questão foram registradas como interpretadas[4].

[4] Respostas interpretadas: Não estão transcritas literalmente como foram respondidas pelos estudantes, mas sim, foi feita uma interpretação geral de suas anotações.

Tabela 8 – Análise da Questão 4 - Teste 1

Grupo participante da célula	
Nº de respostas	**Respostas - interpretadas**
2	Não, um deles não acrescentou nada mais, e o outro afirmou que não conhecia, mas que acredita ser uma ótima forma de ensinar lógica de programação
2	Sim, um ouviu falar através de vídeos, e o outro relatou que já utilizou a ferramenta Construct 2 de programação por blocos, e também considera uma boa forma de ensinar lógica de programação
Grupo não participante da célula	
Nº de respostas	**Respostas - interpretadas**
3	Não, acreditam que seja um tipo de programação "pré-ajustado"que já possui muitas funcionalidades prontas e possibilite a criação aplicativos simples
1	Sim, que se é possível criar jogos simples
Análise: Concordância entre os dois grupos no aspecto de programação por blocos servir majoritariamente para a criação de programas simples. Predominância no grupo participante da célula, o relato da ferramenta App Inventor proporcionar um bom ambiente para o ensino e desenvolvimento de lógica de programação.	

Fonte: Elaborado pelo autor.

Questão 5 - Questão de lógica de programação (Tabela 9).

Tabela 9 – Análise da Questão 5 - Teste 1

Grupo participante da célula	
Nº de respostas	**Respostas**
4	Correta
Grupo não participante da célula	
Nº de respostas	**Respostas**
3	Correta
1	Incorreta
Análise: Pequena diferença entre as respostas dos dois grupos. Com o grupo participante da célula levando uma moderada vantagem.	

Fonte: Elaborado pelo autor.

Questão 6 - Questão que trata da semântica de lógica de programação (Tabela 10).

Tabela 10 – Análise da Questão 6 - Teste 1

Grupo participante da célula	
Nº de respostas	**Respostas**
3	Correta
1	Incorreta
Grupo não participante da célula	
Nº de respostas	**Respostas**
4	Correta
Análise: O grupo não participante obteve melhor resultado. Por se tratar de uma questão que envolve semântica, acredita-se que o único erro ocorrido possa ter sido conta de uma leitura desatenciosa, por parte do participante da célula.	

Fonte: Elaborado pelo autor.

Questão 7 - Questão que trata de pensamento computacional (Tabela 11).

Tabela 11 – Análise da Questão 7 - Teste 1

Grupo participante da célula	
Nº de respostas - não exclusivas	**Respostas - interpretadas**
3	Pensamento relacionado ao planejamento, antes da execução
2	Dedução de que a execução da construção de paredes deveria ser uniforme, e não subdivida
1	Relato citando a possibilidade do uso de uma função
Grupo não participante da célula	
Nº de respostas - não exclusivas	**Respostas - interpretadas**
2	Relato de planejamento antes da execução
1	Se mostrou ter o pensamento literal, sem ligação com o modo pessoal de programar
1	Concordou com a abordagem da construção da casa da forma que está apresentado
Análise: Foi comum aos dois grupos um pensamento relacionado ao planejamento antes da execução. Contudo, o grupo participante da célula apresentou ter mais pensamento crítico, assim como também ocorreu uma reflexão sobre a possibilidade do uso de funções.	

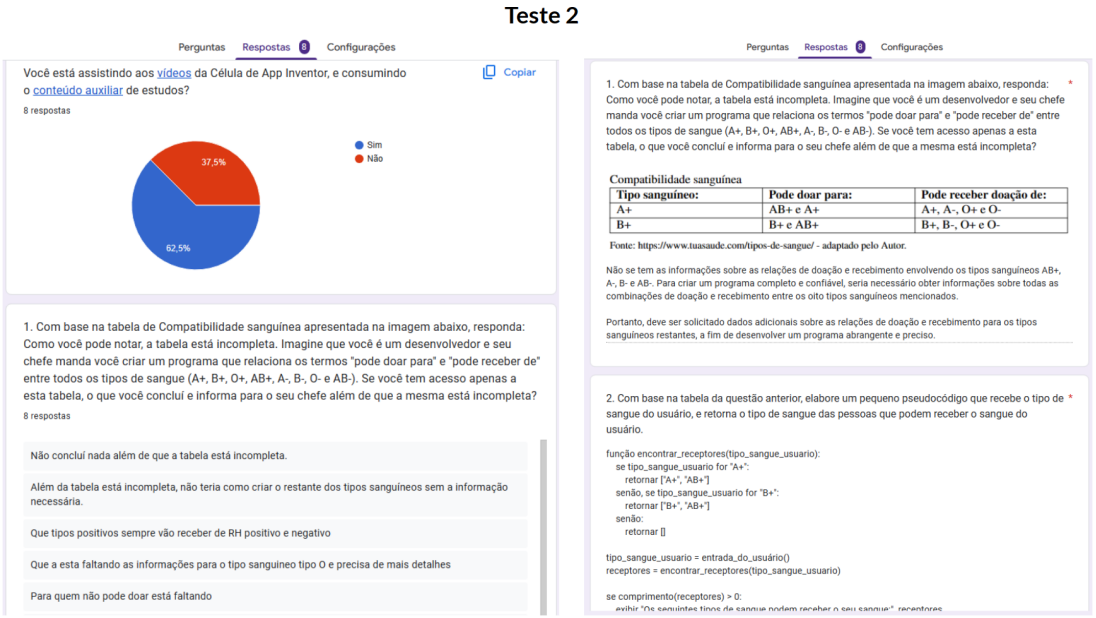
Fonte: Elaborado pelo autor.

5.1.2 Teste 2

A Figura 16 mostra uma prévia do Teste 2 na plataforma Google Forms. Na coleta de respostas do segundo teste foi notado que um dos estudantes que estava no grupo não participante da célula, entrou para o grupo participante. De forma que ele respondeu que estava assistindo as videoaulas dos encontros da célula, assim como consumindo o material auxiliar de estudo.

Logo, este teste conteve 5 respostas do grupo participante da célula, e 3 respostas do grupo não participante da célula. Lembrando que estes são os mesmos indivíduos que fizeram o primeiro teste.

Figura 16 – Prévia do Teste 2 na plataforma Google Forms



Fonte: Elaborado pelo autor.

Conforme as respostas das 5 perguntas que compuseram o segundo teste, os resultados puderam ser categorizados da seguinte forma:

Questão 1 - Interpretação de uma tabela incompleta que define regras de relacionamentos de compatibilidade entre tipos sanguíneos (Tabela 12).

Tabela 12 – Análise da Questão 1 - Teste 2

Grupo participante da célula	
Nº de respostas	**Respostas - interpretadas**
2	Resposta bem construída, concluindo que só seria possível criar tal programa, com o conhecimento de todas as relações de compatibilidade sanguínea
1	Pessoas com tipo sanguíneo positivo sempre irão poder receber doação de tipo sanguíneo positivo e negativo
1	Citou que não há nada sobre o tipo sanguíneo O
1	Não acrescentou nada além de que a tabela está incompleta
Grupo não participante da célula	
Nº de respostas	**Respostas - interpretadas**
1	Acrescentou que não seria possível relacionar todos os tipos de sangue
1	Os dados de restrição de doação de sangue não são apresentados
1	Não soube responder
Análise: Respostas bem estruturadas e justificadas só estão presentes no grupo participante da célula. Porém, nos dois grupos (participante e não participante da célula), houve pelo menos uma conclusão correta, e outra conclusão resultante da análise dos dados que, mesmo que incompletos, estão na tabela.	

Fonte: Elaborado pelo autor.

Questão 2 - Questão para elaborar pseudocódigo simples com estrutura de seleção (Tabela 13).

Tabela 13 – Análise da Questão 2 - Teste 2

Grupo participante da célula	
Nº de respostas	**Respostas**
3	Correta
1	Não soube responder
1	Parcialmente correta
Grupo não participante da célula	
Nº de respostas	**Respostas**
1	Incorreta
1	Correta
1	Parcialmente correta
Análise: Foi maior o número de acertos entre os participantes da célula. Apesar de que, foi neste grupo que um participante nem mesmo tentou elaborar um pseudocódigo qualquer. Podendo evidenciar displicência deste participante ao responder à questão. O resultado do grupo não participante da célula, pôde ser considerado mediano, se comparado ao grupo participante.	

Fonte: Elaborado pelo autor.

Questão 3 - Complementar ao que foi solicitado na questão anterior, também para elaborar pseudocódigo com estrutura de seleção (Tabela 14).

Tabela 14 – Análise da Questão 3 - Teste 2

Grupo participante da célula	
Nº de respostas	**Respostas**
2	Correta
1	Não soube responder
1	Parcialmente correta
1	Foi notado que o participante pesquisou a tabela completa e forneceu o algoritmo completo (fez além do que lhe foi pedido)
Grupo não participante da célula	
Nº de respostas	**Respostas**
1	Incorreta
1	Correta
1	Parcialmente correta
Análise: Os resultados de ambos os grupos se mostraram bem semelhantes aos da Questão 2. Valendo ressaltar que no grupo participante da célula, um dos estudantes respondeu mais do que lhe foi solicitado, mostrando que houve entusiasmo, da sua parte, em entender como ficaria o algoritmo completo para o problema apresentado pelas Questões 2 e 3.	

Fonte: Elaborado pelo autor.

Questão 4 - Analisar trecho de código com erro lógico dentro de uma estrutura de seleção (Tabela 15).

Tabela 15 – Análise da Questão 4 - Teste 2

Grupo participante da célula	
Nº de respostas - não exclusivas	**Respostas - interpretadas**
3	Não identificou o erro no código
2	Informou como corrigir o algoritmo
2	Concluiu que o algoritmo está incorreto
2	Para reduzir o algoritmo: como os dois casos já foram tratados nos dois primeiros se, não precisaria do último se não se, bastaria um se não
1	Para reduzir o algoritmo: pensamento mais alto nível, aplicar o algoritmo em uma função, e chamá-lo sempre que necessário
1	Para reduzir o algoritmo: forneceu uma abordagem com método de ordenação
1	Não informou como diminuir o algoritmo
Grupo não participante da célula	
Nº de respostas - não exclusivas	**Respostas - interpretadas**
2	Não forneceu um método para diminuir o código
2	Informou como corrigir o algoritmo
1	Fez uma interpretação do que é retornado do jeito que o algoritmo está
1	Para reduzir o algoritmo: forneceu uma abordagem com método de ordenação
1	Não identificou o erro no código
Análise: Esta questão talvez foi a que mais mostrou o impacto do segundo teste ter sido aplicado sobre 5 estudantes participantes da célula e apenas 3 estudantes não participantes da célula, tendo uma variedade maior de respostas e interpretações. Contudo, o grupo participante da célula, teve resultado que pôde ser considerado ruim, se comparado com o resultado das outras questões dos dois testes aplicados. Sendo que a maioria dos seus integrantes não perceberam o erro do código, respondendo apenas a segunda e última pergunta feita na Questão, que foi a de como reduzir o algoritmo, mostrando desatenção de leitura, pela maioria destes estudantes. A desatenção na leitura do anunciado desta questão se mostrou ser um problema presente nos estudantes dos dois grupos. O restante destes resultados foram semelhantes, com os dois grupos fornecendo meios de reduzir o pseudocódigo de formas parecidas, assim como formas de corrigi-lo.	

Fonte: Elaborado pelo autor.

Questão 5 - Pensamento computacional sobre um algoritmo não fornecido pela questão, com estrutura de repetição e seleção (Tabela 16).

Tabela 16 – Análise da Questão 5 - Teste 2

Grupo participante da célula	
Nº de respostas	**Respostas - interpretadas**
2	Resposta superficial, com while e if, sem apresentar a lógica da resolução do problema
2	Correta, utilizando o módulo do número por 10 e fatorando por 10, comparando o resultado com o valor procurado, até o número chegar em 0
1	Fora de contexto, com o uso de string ou char
Grupo não participante da célula	
Nº de respostas	**Respostas - interpretadas**
3	Fora de contexto, com o uso de string ou char
Análise: Resultados semelhantes, podendo ressaltar o melhor desempenho do grupo participante da célula. Pois somente nele houve respostas corretas, conforme foi solicitado pela Questão. Apesar de que as respostas dos demais participantes dos dois grupos, não foram completamente erradas, apenas não seguiram a forma como foi pedido na Questão (esta que exigia mais da construção lógica do algoritmo).	

Fonte: Elaborado pelo autor.

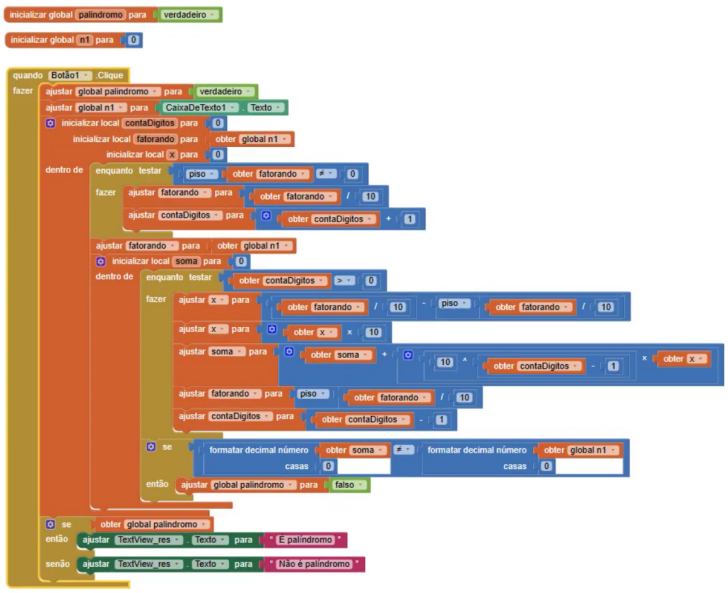
5.2 Conclusão da célula

Para finalizar a célula, no vídeo[5] do último encontro foi mostrado aos participantes um aplicativo feito com React Native, por se tratar de uma aplicação um pouco complexa, já com front-end, back-end, e várias telas, ela teve apenas uma tela comparada, de forma simplificada, a tela de um aplicativo feito com o App Inventor criado previamente na célula. A Figura 17 mostra uma prévia dos códigos dos dois aplicativos comparados.

[5] Link para do vídeo de conclusão da célula: https://youtu.be/G7yh1tgp_GY

Figura 17 – Prévia da comparação feita no vídeo de conclusão da célula dos dois aplicativos.

Feito com App Inventor



Feito com React Native

Fonte: Elaborado pelo autor.

Esta etapa foi útil para evidenciar a quantidade de funcionalidades que o App Inventor e a programação por blocos proporcionam aos estudantes, simplificando processo de desenvolvimento de aplicativos para programadores iniciantes. A Figura 18 exibe quatro aplicativos criados durante os encontros da célula, lembrando que todos os aplicativos podem ser encontrados no material de estudos suplementar disponível no GitHub.

Figura 18 – Alguns aplicativos criados durante a célula.

Fonte: Elaborado pelo autor.

6 CONCLUSÃO

Este trabalho teve o intuito de aplicar a utilização da ferramenta App Inventor no ensino de programação em cursos de graduação na área de TI, verificando também se a programação por blocos causou melhoria nas habilidades de lógica de programação e pensamento computacional dos estudantes que consumiram os materiais criados por este projeto. De forma a realizar um experimento, por meio de uma célula de estudos, com dois grupos de estudantes. Estes que foram submetidos a testes aplicados de forma remota, visando colher suas impressões e entendimento dos conteúdos de fundamentos de programação. Esta célula de estudos foi realizada por meio de vídeos publicados no YouTube e por um material auxiliar de estudos publicado no GitHub, ambos elaborados e executados pelo autor deste trabalho.

No Capítulo 5, estão listadas todas as respostas dos estudantes que fizeram parte do experimento realizado neste trabalho. Tal resultado foi organizado e separado por questões, colocado em tabelas, contendo também uma análise da comparação entre as respostas lá presentes.

Com base nos resultados e na análise geral do Teste 1, foi percebido que de modo geral, os estudantes participantes da célula de estudos apresentaram ter um interesse maior por programação do que os estudantes do grupo não participante da célula de estudos, o que já era esperado. Foi notado também que programação por blocos - forma de programação do App Inventor - não é conhecida pela maioria dos estudantes, e entre os que a conhecem, acreditam que ela serve apenas para a criação de jogos e programas simples.

Sobre as respostas das questões que tratavam mais especificamente de lógica de programação do Teste 1, pôde-se concluir resultados semelhantes entre os dois grupos analisados. Com ressalva sobre a questão em aberto sobre pensamento computacional, onde o grupo participante da célula apresentou ter um pensamento crítico um pouco melhor elaborado, demonstrando maior familiaridade com desenvolvimento de algoritmos, onde houve respostas do tipo: *isso podeira ter sido posto dentro de uma função*, por exemplo.

Partindo para o Teste 2, este que conteve apenas questões mais voltadas a lógica de programação e pensamento computacional. A primeira coisa que vale ressaltar foi que um aluno do grupo não participante da célula, se interessou em consumir os conteúdos deste trabalho. Como o objetivo deste projeto também é propagar conhecimento, este estudante não foi restringido de fazer isso. O restante das respostas que não foi listado no Capítulo 5 pode ser encontrado em forma transcrita no anexo A deste trabalho.

Apesar da diferença entre número de participantes nos grupos no segundo teste, foi possível se concluir que o grupo consumidor dos materiais do YouTube e GitHub apresentou bons resultados, no que diz receito a número de acertos e erros das questões. Porém, não tão distantes do grupo que não consumiu este material, se analisados de maneira proporcional.

Contudo, sobre o aspecto da capacidade de interpretação e construção de algumas respostas, tirando algumas poucas respostas, foi percebido que os participantes do grupo de estudos se saíram melhor. De modo a explicarem melhor suas respostas, estas que continham deduções resultantes de uma capacidade de abstração de algoritmos um pouco melhor desenvolvida, se comparada as respostas do grupo não participante da célula.

Portanto, de modo geral a experiência em usar o App Inventor neste projeto foi bastante interessante, tanto do ponto de vista acadêmico, onde a análise da comparação entre grupos de estudantes pôde proporcionar uma analogia com a obtenção de conhecimento a partir de uma estrutura semelhante à metodologia científica. Quanto pessoal, onde foi percebido pelo autor um gosto por ensinar programação, e pela disseminação de conhecimento.

6.1 Considerações Finais

Por fim, fica também registrado as limitações deste trabalho, onde ao estimular o desenvolvimento de aplicativos móveis, não a apresentou, ou apresentou apenas superficialmente conceitos essenciais para uma construção mais completa de tais aplicativos. Como, por exemplo, conceitos de programação orientada a objetos, controle de eventos, além de back-end e front-end. Também por se tratar de um experimento aberto (sujeito a mudança e adaptação, como de fato ocorreu), deve servir apenas como material intelectual útil para reflexão sobre o uso do App Inventor no ensino programação, que por este trabalho se mostrou, sim, ser satisfatório.

Alguns aprendizados que este trabalho deixou são:

1. Divulgação online é insuficiente para realização de um experimento do tipo do qual foi realizado por este trabalho.
2. Baixa adesão devido a possível preconceito com programação por blocos por parte dos estudantes de graduação.

6.2 Trabalhos Futuros

Em vista das limitações que este trabalho possuiu, ficam algumas sugestões de atitudes que podem ajudar possíveis trabalhos futuros com temas parecidos em sua coleta de resultados:

1. Convidar professores de design para incentivarem suas turmas a participarem da pesquisa.
2. Convidar servidores do campus para participarem da pesquisa.

REFERÊNCIAS

ASCENCIO, A. F. G.; CAMPOS, E. A. V. d. **Fundamentos da Programação de Computadores**: Algoritmos, pascal, c, c++ e java. 3. ed. São Paulo, SP, Brasil: Pearson Universidades, 2012.

BENETTI, P. C.; SOUSA, A. I.; SOUZA, M. H. do N. **Creditação da extensão universitária nos cursos de graduação: relato de experiência**. [*S. l.*]: Revista Brasileira de Extensão Universitária, 2015. v. 6. 25–32 p.

BOSSE, Y.; GEROSA, M. A. **Why is programming so difficult to learn? Patterns of Difficulties Related to Programming Learning Mid-Stage**. New York, NY, USA: ACM, 2017. v. 41. 1–6 p.

BRACKMANN, C. P. **Desenvolvimento do pensamento computacional através de atividades desplugadas na educação básica**. [*S. l.*: *s. n.*], 2017.

CAMARGO, L. L. **Cresce procura por cursos de graduação em Ciência da Computação.** Agência de Conteúdo DN. Jornal Diário do Nordeste, Fortaleza. 2021. Disponível em: https://diariodonordeste.verdesmares.com.br/metro/cresce-procura-por-cursos-de-graduacao-em-ciencia-da-computacao-1.3070288. Acesso em: 31 de out. 2022.

FAUSTINO, G. K. d. S.; CALAZANS, H. K. N. d. S.; LIMA, W. D. **Android e a influência do Sistema Operacional Linux**. [*S. l.*]: Tecnologias em Projeção, 2017. v. 8. 100–111 p.

FORTUNATO, D.; BERNARDINO, J. **Progressive web apps: An alternative to the native mobile Apps**. Carceres, Espanha: IEEE, 2018. 1–6 p.

GIORDAN, I. **50 melhores faculdades de Ciência da Computação, segundo o MEC. Quero Bolsa**. 2021. Disponível em: https://querobolsa.com.br/revista/50-melhores-faculdades-de-ciencia-da-computacao-segundo-o-mec. Acesso em: 1 de nov. 2022.

GOMES, T. C.; MELO, J. C. de. **App inventor for android: Uma nova possibilidade para o ensino de lógica de programação**. Recife, PE, Brasil: CBIE, 2013. v. 2.

GUIMARÃES, L. J. B. L. S.; ARRUDA, A. P. D.; MARTINS, A. L. **O uso do software livre App Inventor no processo de ensino e aprendizagem da lógica no curso de graduação em Sistemas de Informação**. Belo Horizante, MG, Brasil: UEADSL, 2016. v. 1.

RODRIGUES, R. K. **Estudo do uso da linguagem de blocos scratch no ensino do pensamento computacional**. Dissertação (B.S. thesis) – Universidade Tecnológica Federal do Paraná, 2019.

SILBERSCHATZ, A.; GAGNE, G.; GALVIN, P. B. **Operating System Concepts**. 10. ed. Kendallville, Indiana, United States of America: Wiley, 2018.

SOUSA, L.; FARIAS, E.; CARVALHO, W. **Programação em Blocos Aplicada no Ensino do Pensamento Computacional: Um Mapeamento Sistemático**. Porto Alegre, RS, Brasil: SBC, 2020. 1513–1522 p.

VERMA, N.; KANSAL, S.; MALVI, H. **Development of native mobile application using android studio for cabs and some glimpse of cross platform apps**. [*S. l.*]: International Journal of Applied Engineering Research, 2018. v. 13. 12527–12530 p.

VIEIRA, R. S. dos S.; FURTADO, M. da C.; SILVA, A. F. da; VIEIRA, D. S. dos S. **O Uso da Plataforma MIT App Inventor no Incentivo a Inserção de Alunos do Ensino Médio ao Ensino Superior**. [*S. l.*]: Brazilian Journal of Development, 2020. v. 6. 76523–76527 p.

WOLBER, D.; ABELSON, H.; SPERTUS, E.; LOONEY, L. **App inventor**. Sebastopol, California, USA: O'Reilly Media, Inc., 2011.

YASWINSKI, M. R.; CHOWDHURY, M. M.; JOCHEN, M. **Linux security: a survey**. East Stroudsburg, PA, USA: 2019 IEEE International Conference on Electro Information Technology (EIT), 2019. 357–362 p.

APÊNDICE A – TESTES DE MOTIVAÇÃO E CONHECIMENTO

Teste 1

1. Em uma escala de 1 a 7, onde 1 é "nada motivado", 4 é "tanto faz", e 7 é "muito motivado", o quão motivado estou a, no futuro, me especializar na área de programação?
2. Antes de entrar na universidade, eu já tinha contato com programação de alguma forma?
3. Até agora, qual subárea da ciência da computação em geral eu acho mais interessante de se estudar?
4. Anteriormente, eu já tinha ouvido falar sobre programação por blocos?
   Se sim, o que vi se criar com ela?
   Se não, o que imagino que seja possível se criar com ela?
5. Qual o valor de retorno do trecho de código a seguir:

   <início>
   inicializa n1 = 4, n2 = 6
   se n1 < n2-(0.1*n2) faça:
       retorne n2 - n1
   se não faça:
       retorne n1 - n2
   <fim>
6. O que está errado no trecho de código a seguir:

   - Algoritmo de cálculo de desconto:
   <início>
   inicializa valor = 70, descontoEmPorcento = 15
   retorne valor-descontoEmPorcento
   <fim>
7. A seguir temos um algoritmo referente a logística do processo de construção de uma casa:

   <início>
   comprar material de construção
   contratar um pedreiro
   fazer o alicerce
   levantar paredes da sala de estar
   levantar paredes da cozinha
   levantar paredes do primeiro quarto

levantar paredes do segundo quarto

...

<fim>

Este algoritmo parece correto? É necessário mudar algo nele? Se sim, o que.

Teste 2

1. Analise tabela a seguir:

Compatibilidade sanguínea

Tipo sanguíneo:	**Pode doar para:**	**Pode receber doação de:**
A+	AB+ e A+	A+, A-, O+ e O-
B+	B+ e AB+	B+, B-, O+ e O-

Fonte: https://www.tuasaude.com/tipos-de-sangue/ - adaptado pelo Autor.

Como você pode notar, a tabela está incompleta. Imagine que você é um desenvolvedor e seu chefe manda você criar um programa que relaciona os termos "pode doar para"e "pode receber de"entre todos os tipos de sangue (A+, B+, O+, AB+, A-, B-, O- e AB-). Se você tem acesso apenas a esta tabela, o que você concluí e informa para o seu chefe?

2. Com base na tabela da questão anterior, elabore um pequeno pseudocódigo que recebe o tipo de sangue do usuário, e retorna o tipo de sangue das pessoas que podem receber o sangue do usuário.
3. Com base na tabela da Questão 1, faça o contrário da questão anterior, projete um pequeno pseudocódigo que recebe o tipo de sangue do usuário, e retorna quais são os tipos sanguíneos que ele pode receber doações de sangue, caso ele precise algum dia.
4. Tem algo errado no algoritmo a seguir? Justifique sua resposta.

- Algoritmo que recebe 3 inteiros positivos e distintos, e ao considerar os números em ordem, retorna o número do meio. Exemplo: n1 = 10, n2 = 5, e n3 = 15, deve retornar n1.

<início>

leia n1, n2, n3

se n1 > n2 ou n1 < n3 faça:

    retorne n1

se não se n2 > n1 ou n2 < n3 faça:

    retorne n2

se não se n3 > n1 ou n3 < n2 faça:

    retorne n3

<fim>

5. Imagine um algoritmo que recebe como entrada um algarismo de 0-9 ’n’, e um número de celular ’c’, e retorna a quantidades de vezes que ’n’ aparece em ’c’. Considerando a restrição do algoritmo não poder usar Strings, e sim trabalhar sempre com números. Quais estruturas de controle e/ou repetição você acha que são necessárias para resolver este problema? Escreva um pouco sobre sua escolha.

ANEXO A – RESTANTE DAS RESPOSTAS DOS ESTUDANTES COLETADAS PELOS TESTES DESTE PROJETO

TESTE 1

Questão 4

- Não
- Não, criar aplicativos, uma otima forma para ensinar
- Sim, em vídeos
- Sim, sei que é possível criar jogos simples através da programação por blocos.
- Não. Programas de forma mais simplificada
- Não. Creio que seja uma maneira de programar mais pré-disposta , já tendo um bloco de código preparado para cada etapa.
- Sim, utilizei no Construct 2, um programa que permite criar jogos construindo a lógica do jogo sem usar linguagem de programação, apenas com um sistema de encaixar a lógica. Também já tinha visto programação por blocos com o intuito de ensinar lógica.

Questão 6

- O sinal de subtração deveria ser de multiplicação
- Ele teria que subtrair a variável valor pelo resultado obtido através da multiplicação da variável valor pela variável descontoEmPorcento e após a multiplicação fazer a divisão por 100, para assim poder efetuar corretamente o cálculo.
- O valor que está sendo retornado é apenas uma subtração de números inteiros, não tem nada relacionado a porcentagem.
- O cálculo do desconto,o certo seria fazer (((100-15) / 100) * 70,dessa forma já retornaria o valor,descontado o desconto
- É necessário criar uma variável que calcule o valor do desconto e em seguida subtraia esse valor do preço original.
- A variavel “descontoEmPorcento” não está sendo tratado da forma correta, levando a um erro de cálculo devido a subtração da valor pelo desconto como um inteiro. O certo seria fazer uma regra de porcentagem para tratá-la de forma adequada
- Faltou transformar o desconto em porcento em dinheiro para subtrair do valor inicial.
- o algoritmo deveria retornar o valor menos 15% de desconto em relação ao próprio valor 70. algo mais ou menos assim: retorne valor*1-(15/100)

Questão 7

- Sim esta certo, se desconsiderar o fator da casa ter a necessidade de construir o piso e o contra-piso
- Eu levantaria os muros externos da casa primeiro e colocaria as vigas e modelava antes de sair levantando um por um, mas como não entendo muito é isso
- Acrescentaria passos como: definir o terreno, preparar massa. E todas as paredes podem ser erguidas de forma independente de cômodo
- Fazer um planejamento
- Aparentemente está correto.
- Talvez falte alguns detalhes antes do levantamento das paredes como a construção de colunas por exemplo
- Parece incompleto. Faltou o banheiro.
- É possível seguir essa lógica em uma construção, nada impede que cada cômodo tenha suas paredes levantadas por vez antes de cobrir a casa, mas seria recomendado levantar todas as paredes de uma vez só para uma melhor eficiência. Mas os passos de levantar as paredes, poderiam serem colocados dentro de uma função que recebesse o parâmetro para cada cômodo.

TESTE 2

Questão 1

- Não concluí nada além de que a tabela está incompleta.
- Além da tabela está incompleta, não teria como criar o restante dos tipos sanguíneos sem a informação necessária.
- Que tipos positivos sempre vão receber de RH positivo e negativo
- Que a esta faltando as informações para o tipo sanguineo tipo O e precisa de mais detalhes
- Para quem não pode doar está faltando
- Não se tem as informações sobre as relações de doação e recebimento envolvendo os tipos sanguíneos AB+, A-, B- e AB-. Para criar um programa completo e confiável, seria necessário obter informações sobre todas as combinações de doação e recebimento entre os oito tipos sanguíneos mencionados. Portanto, deve ser solicitado dados adicionais sobre as relações de doação e recebimento para os tipos sanguíneos restantes, a fim de desenvolver um programa abrangente e preciso.
- Não sei
- que o sistema iria informar somente com base o que está na tabela, o que seria bem

limitado, eu não poderia criar informações que não foram me apresentadas.

Questão 2

- Não sei
- TipoSangue(valor)

  Se valor for igual A+

  Retorne “A+, A-, O+, O-“

  Se valor for igual B+

  Retorne “B+, B-, O+, O-“
- Se(A+)

  Return (AB+ e A+)

  Se então (B+)

  Return (B+ e AB+)
- inseri tipoUsuario

  Se tipoUsuario == tipoSanguineoA

  retorna tiposPodeReceberDeA

  Se tipoUsuario == tipoSanguineoB

  retorna tiposPodeReceberDeB

  Se tipoUsuario == tipoSanguineoO

  retorna tiposPodeReceberDeO
- Ler tipodesangue

  For(i=0 ;i < [vetor de possibilidade] ; i++)

  Exibir tipo-de-sangue-que-pode-receber
- função encontrar_receptores(tipo_sangue_usuario):

  se tipo_sangue_usuario for "A+":

  retornar ["A+", "AB+"]

  senão, se tipo_sangue_usuario for "B+":

  retornar ["B+", "AB+"]

  senão:

  retornar []

  tipo_sangue_usuario = entrada_do_usuário()

  receptores = encontrar_receptores(tipo_sangue_usuario)

  se comprimento(receptores) > 0:

exibir "Os seguintes tipos de sangue podem receber o seu sangue:", receptores

senão:

exibir "Não há informações suficientes para determinar os receptores para o seu tipo de sangue."

- sangue = str(input('Digite seu tipo sanguíneo: '))

  if sangue in 'A+':

  print(' As pessoas com os tipos sanguíneos: AB+ e A+, podem receber sangue do usuario.')

  else:

  if sangue in 'B+':

  print(' As pessoas com os tipos sanguíneos: B+ e AB+, podem receber sangue do usuario.')
- se tipoSanguineo == A+: retorna "AB+ e A+"

  se não se tipoSanguineo == B+: retorna "B+ e AB+"

Questão 3

- Não sei
- TipoSangue(valor)

  Se valor for igual A+

  Retorne "A+, AB+"

  Se valor for igual B+

  Retorne "B+, AB+"
- Se(A+)

  Return (A+, A-, O+ e O-)

  Se então (B+)

  Return (B+,B-, O+ e O-)
- inseri tipoUsuario

  Se tipoUsuario == tipoSanguineoA

  retorna tiposPodeDoarParaA

  Se tipoUsuario == tipoSanguineoB

  retorna tiposPodeDoarParaB

  Se tipoUsuario == tipoSanguineoO

  retorna tiposPodeDoarParaO
- Ler tipodesangue

  For(i=0 ;i < [vetor de possibilidade] ; i++)

Exibir tipo-de-sangue-compativeis

- função encontrar_doadores(tipo_sangue_usuario):
  se tipo_sangue_usuario for "A+":
  retornar ["A+", "A-", "O+", "O-"]
  senão, se tipo_sangue_usuario for "B+":
  retornar ["B+", "B-", "O+", "O-"]
  senão, se tipo_sangue_usuario for "O+":
  retornar ["O+", "O-"]
  senão, se tipo_sangue_usuario for "AB+":
  retornar ["A+", "A-", "B+", "B-", "AB+", "AB-", "O+", "O-"]
  senão, se tipo_sangue_usuario for "A-":
  retornar ["A-", "O-"]
  senão, se tipo_sangue_usuario for "B-":
  retornar ["B-", "O-"]
  senão, se tipo_sangue_usuario for "O-":
  retornar ["O-"]
  senão, se tipo_sangue_usuario for "AB-":
  retornar ["A-", "B-", "AB-", "O-"]
  senão:
  retornar []
  tipo_sangue_usuario = entrada_do_usuário()
  doadores = encontrar_doadores(tipo_sangue_usuario)
  se comprimento(doadores) > 0:
  exibir "Você pode receber doações dos seguintes tipos de sangue:", doadores
  senão:
  exibir "Não há informações suficientes para determinar os doadores para o seu tipo de sangue."
- sangue = str(input('Digite seu tipo sanguíneo: '))
  if sangue in 'A+':
  print(' As pessoas com os os tipos sanguíneos: A+, A- , O+ e O-, podem receber sangue do usuario.')
  else:

if sangue in ’B+’:

print(’ As pessoas com os os tipos sanguíneos: B+, B- , O+ e O-, podem receber sangue do usuario.’)

– se tipoSanguineo == A+: retorna "A+, A-, O+ e O-"

se não se tipoSanguineo == B+: retorna "B+, B-, O+ e O-"

Questão 4

– Está correto

– O algoritmo está incorreto, pois se considerar que o algoritmo deveria retornar apenas um valor dos 3 lidos, o OR no condicional if, ele vai imprimir os 3 valores sempre. O correto seria usar um AND, assim na primeira condição falsa ele não retornaria nenhum valor. Uma forma de diminuir o código seria ordenando os valores com uma função que a linguagem oferece e retornando o valor na posição 1 do array, por exemplo

– O último se não se, poderia se apenas um se não. Pois ou outros casos já foram estabelecidos.

– Criaria uma função para realizar essa comparação e chamaria toda vez que necessario

– Trocar ou por e

– Sim, há um erro no algoritmo apresentado. O problema ocorre nas condições de verificação dentro do bloco "se"e "senão se". A lógica utilizada não está correta para determinar o número do meio. Uma maneira correta de implementar o algoritmo é ordenar os três números em ordem crescente e retornar o número do meio. Uma possível solução seria a seguinte:

less

Copy code

<início>

leia n1, n2, n3

se n1 > n2, faça:

troque n1 e n2

se n2 > n3, faça:

troque n2 e n3

se n1 > n2, faça:

troque n1 e n2

retorne n2

<fim>

Essa solução consiste em utilizar as trocas condicionais para garantir que os números sejam ordenados corretamente. Após a ordenação, o número do meio (n2) é retornado.

- não consigo pensar em outra forma de fazer o algoritmo no momento
- Sim, o algoritmo está errado, ele deveria usar a condição E, pois se o n1 for maior que o n2 ou menor que o n3, ele irá retornar o n1 se apenas uma das condições forem atendidas, o que não garante que o n1 sempre será o número do meio. Utilizando a condição E, o código também poderia ser diminuido, onde ele deveria verificar apenas se o N1 é maior que o N2 e Menor que o N3 para retornar o N1 e também se o N2 é maior que o N1 e menor que o N3, se ele não satisfazer nenhuma dessas condições, significa que o N3 é o valor do meio.

  leia n1,n2,n3

  se n1 > n2 e n1 < n3:

  retorne n1

  se não se n2 > n1 e n2 < n3:

  retorne n2

  retorne n3

Questão 5

- While. Pois retorna a quantidade de vezes que n aparece em C.
- Ordena de forma crescente o valor c do celular, faz uma comparação com este valor e o valor n de busca e acrescenta um contador para cada vez que o valor “n” foi chamado neste c do celular. For fim retorne o valor do contador
- If e for. São suficientes, já que o número de repetições é conhecido, e não precisa de mais que um If para realizar comparações
- Usaria operações binarias e bitwise, que traz a possibilidade de poder modificar a variavel mudando a posição dos numeros
- Switch,for com se e else,em ambos eu trabalharia com nuneros,mas eles sendo na cerdade do tipo char
- Para resolver o problema de contar a quantidade de vezes que um algarismo ’n’ aparece em um número de celular ’c’, sem utilizar strings e trabalhando apenas com números, uma estrutura de repetição é necessária. A escolha mais adequada para esse caso seria o uso de um laço de repetição, como o laço while ou o laço for. O laço de repetição seria utilizado

para percorrer cada dígito do número de celular ’c’ e compará-lo com o algarismo ’n’. A cada iteração, o algoritmo verificará se o dígito atual é igual ao algarismo ’n’ e, caso seja, incrementará um contador. A escolha do laço de repetição é apropriada porque permite percorrer os dígitos do número de celular um a um, sem a necessidade de utilizar strings. O algoritmo poderá extrair o último dígito do número utilizando a operação módulo (%) e depois removê-lo dividindo o número por 10. Aqui está um exemplo de pseudocódigo que utiliza um laço de repetição while para contar a quantidade de vezes que o algarismo ’n’ aparece no número de celular ’c’:

kotlin

Copy code

função contar_algarismo(n, c):

contador = 0

enquanto c for diferente de zero:

dígito = c % 10 // obtém o último dígito do número

c = c // 10 // remove o último dígito do número

se dígito for igual a n:

contador = contador + 1

retornar contador

Esse algoritmo percorre todos os dígitos do número de celular ’c’ até que não haja mais dígitos. A cada iteração, verifica se o dígito é igual ao algarismo ’n’ e incrementa o contador, caso seja verdadeiro. No final, o algoritmo retorna o contador, que representa a quantidade de vezes que o algarismo ’n’ aparece no número de celular ’c’.

- acredito que a melhor maneira de resolver este problema seria utilizando listas(conhecida como vetores em outras linguagens), para verificar quantas vezes um determinado número aparece seria necessário usar o while ou for, para pecorrer a lista e encontrar o número desejado, para assim contar quantas vezes ele se repete.
- eu usaria um loop while, e eu iria dividindo o numero por 10 até o valor de C ficar igual a zero, para cada repetição, por exemplo o código abaixo:

  n = 3

  c = 993034433

  contador = 0

  enquanto c != 0:

digito = c

se digito == n:

contador += 1

c = c/10

imprime(contador)

assim, a cada iteração do loop, ele sempre iria pegar o ultimo digito utilizando o c%10, e iria diminuir uma casa decimal do numero, o que faria ele pegar o novo ultimo digito, fazendo assim com todos os numeros da direita pra esquerda, do 3 até o 9. e no final printaria o total de vezes que o numero n apareceu no telefone c.