Carpilator: Jogo digital educativo para ensino de programação

Aiesa Moraes Rosa[1], João Vitor Xavier[2], Leonardo Bravo Estácio[3]

[1]Instituto Federal de Santa Catarina - Câmpus Lages
R. Heitor Villa Lobos, 225 - São Francisco, Lages - SC, 88506-400
Curso de Ciência da Computação

aiesamoraesrosa@gmail.com, fastjonh@gmail.com, leonardo.bravo@ifsc.edu.br

Resumo. *A evasão de cursos superiores na área da tecnologia são muito comuns devido a dificuldade que de abstração dos discentes em matérias voltadas a programação, dessa forma jogos digitais têm sido usados como estratégia de gamificação voltado ao contexto educacional para o ensino-aprendizagem de programação. O objetivo deste artigo foi desenvolver um jogo digital para auxiliar o professor no ensino de programação em sala de aula. O trabalho teve início com uma pesquisa de campo, seguindo por uma apresentação dos passos necessários para o desenvolvimento do jogo digital. Os resultados encontrados foram satisfatórios e o jogo produzido poderá auxiliar um docente ao ministrar conteúdos introdutórios de programação.*

Abstract. *Dropping out of higher education courses in the area of technology is very common due to the difficulty that students have in abstracting subjects related to programming, thus digital games have been used as a gamification strategy aimed at the educational context for the teaching-learning of programming. The objective of this article was to develop a digital game to assist the teacher in teaching programming in the classroom. The work began with a field survey, followed by a presentation of the necessary steps for the development of the digital game. The results found were satisfactory and the game produced can help a teacher to teach introductory programming content.*

1. Introdução

A disciplina de programação faz parte da grade curricular, principalmente, dos cursos de graduação de tecnologia, como por exemplo Sistemas de Informação, onde um em cada três alunos ingressantes finalizam o curso e Ciência da Computação, no qual um a cada quatro se forma. É possível pontuar que a falta de compreensão do raciocínio lógico da programação é uma das causas de reprovação e desistência (Silva et al., 2018a).

Segundo Grossmann (2019) a procura por profissionais na área de Tecnologia da Informação (TI) no país será de 420 mil pessoas até o ano de 2024, entretanto, o número se contrapõe à baixa quantidade na formação de profissionais por ano e desperta um alerta para o risco de um apagão de pessoas qualificadas. Um dos fatores que podem causar isso, é porque o processo de aprendizagem dos conceitos iniciais de programação é complexo e precisa de um nível de abstração que não está presente nos discentes que estão tendo um primeiro contato com a área da programação e assim, acabam desanimando e evadindo (Zanetti e Oliveira, 2015).

A evasão no ensino superior, é o ato de desistir e trancar o curso, dessa forma, para diminuir a evasão dos alunos, técnicas têm sido desenvolvidas, uma dessas estratégias é a gamificação, uma técnica que usa a competitividade para atingir um determinado objetivo (Silva et al., 2018a).

Para tentar resolver o problema supracitado, o objetivo deste trabalho é desenvolver um jogo digital para auxiliar o professor no ensino de programação em sala de aula. O nome *Carpilator* vem da mistura de carro em inglês ”*car*”e do substantivos em inglês para aquele que faz a compilação, ”*compilator*”. ”Car”+ ”compilator” = Carpilator.

São objetivos específicos:

- Realizar uma pesquisa com os alunos de graduação que cursaram disciplinas de programação no Instituto Federal de Santa Catarina (IFSC) - Câmpus Lages a fim de verificar as dificuldades ao ter o primeiro contato com o conteúdo;
- Elaborar o design e desenvolver o novo interpretador do jogo digital;
- Testar com alunos de graduação do IFSC - Câmpus Lages o jogo digital desenvolvido e apresentar um relatório com os dados levantados.

Além desta seção introdutória, este documento é composto das seguintes seções: A seção 2 apresenta a metodologia; A seção 3 mostra o referencial teórico necessário para ao entendimento do trabalho; A seção 4 registra o desenvolvimento do trabalho; E, na seção 5, as conclusões sobre o trabalho são apresentadas.

2. Metodologia

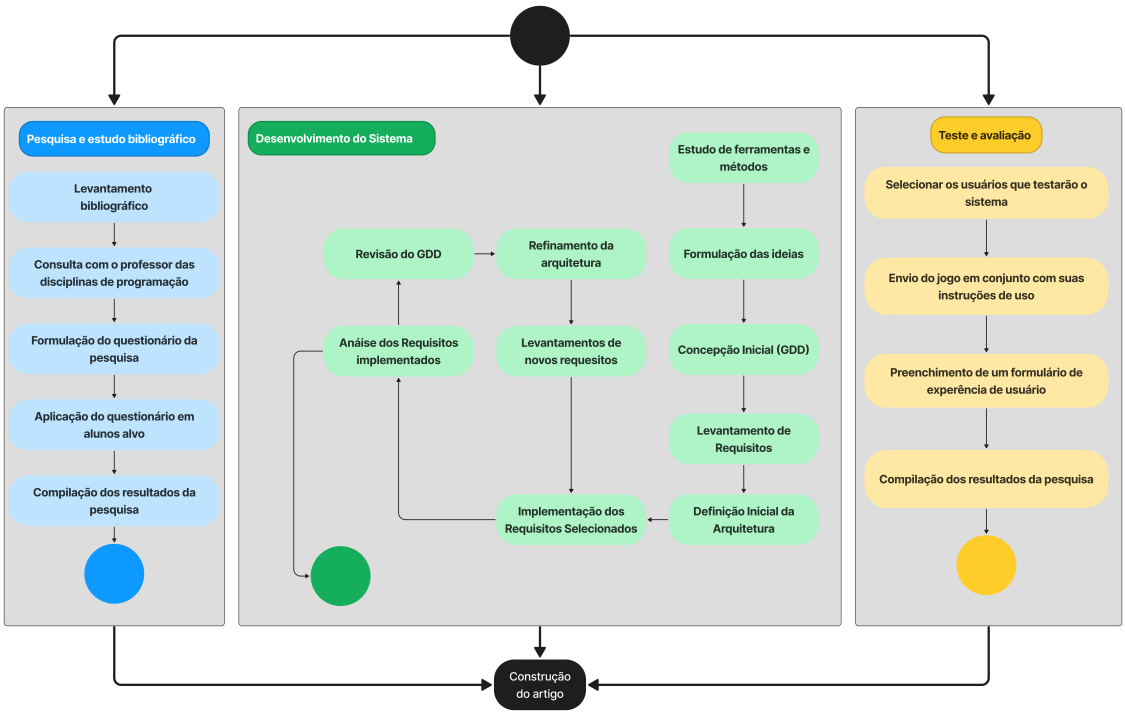

Figura 1. Metodologia geral

Conforme apresentado na Figura 1, o trabalho será composto por 3 etapas. A primeira etapa consiste em uma pesquisa de campo. Já a segunda etapa na apresentação dos passos necessários para o desenvolvimento do jogo digital. Por fim, a terceira etapa consiste no teste de qualidade do jogo desenvolvido nos critérios de usabilidade e funcionalidade e a confecção de um relatório apontando as experiências dos alunos envolvidos.

A primeira etapa consiste em realizar um questionário de múltipla escolha visando identificar os principais problemas apontados pelos alunos de graduação, e, a partir daí, formular estratégias para serem adicionadas ao jogo digital para suprir essas dificuldades. O questionário será formulado junto ao professor da disciplina de Introdução a Programação do IFSC Câmpus Lages.

A segunda etapa consiste no desenvolvimento do jogo digital na plataforma *Unity*, conhecida como motor de jogos que possui como linguagem de programação o C#. Por fim, na terceira etapa o jogo digital estará pronto para ser testado pelos alunos do IFSC - Câmpus Lages e a partir destes testes será elaborado um relatório com as considerações e experiência dos testantes.

Do ponto de vista da natureza, esse trabalho é uma pesquisa aplicada. Quanto à abordagem do problema a pesquisa é qualitativa. Referente aos objetivos, esse artigo classifica-se

como sendo uma pesquisa exploratória. Considerando os procedimentos técnicos, a pesquisa é bibliográfica.

3. Referencial teórico

A seção apresenta os assuntos fundamentais para o entendimento do trabalho como um todo. Estando dividida em 2 subseções. A subseção 3.1 descreve a revisão sistemática, a 3.2 os trabalhos correlatos e a subseção 3.3 descreve as tecnologias utilizadas para o desenvolvimento do jogo digital.

3.1. Revisão sistemática

Para pesquisar trabalhos semelhantes a este projeto foi realizada uma revisão sistemática na literatura. A ferramenta utilizada foi o *Google Acadêmico* através da *string* de consulta "Jogos digitais voltados para o ensino da programação". Foram selecionados os 20 primeiros artigos com período de tempo como parâmetro de 8 anos para serem analisados. Após a leitura dos resumos e objetivos foi atribuído um valor de relevância, de 1 a 5, para cada artigo, sendo 1 pouco relevante e 5 muito relevante. Dos 20 trabalhos selecionados, foi feita uma leitura completa nos 8 de maiores notas e desses foram escolhidos os 4 trabalhos de maior relevância e importância para serem resumidos nesse artigo.

3.2. Trabalhos correlatos

O trabalho de Andrade et al. (2016) descreve, uma abordagem que visa estimular a programação no ensino médio por meio do desenvolvimento de jogos digitais, dando suporte à multidisciplinaridade intrínseca ao processo, através da ferramenta *Scratch*. Foi estimulada a competitividade entre os grupos e os resultados apontaram que os alunos conseguiram documentar o processo de produção, desenvolveram habilidades cognitivas relacionadas à programação e empregaram heurísticas de jogabilidade nos jogos produzidos.

O projeto proposto por Silva et al. (2018b) tem como foco replicar uma revisão sistemática da literatura (RSL) referente aos anos de 2008 a 2012, e apresentar uma nova RSL sobre o uso de jogos digitais focado no ensino de programação, para iniciantes em Computação no nível superior, no Brasil abrangendo 2013-2017. Foram comparados os resultados obtidos em ambos os períodos da última década e identificadas as ferramentas e teorias pedagógicas que são aplicadas para o ensino-aprendizagem de programação.

Monclar et al. (2018) realizaram uma revisão, iniciando no ano de 2001 até 2018, buscando identificar jogos que promovam a evolução dos discentes na disciplina de programação no ensino superior. Também foram considerados aqueles que fundamentavam os pilares do pensamento computacional, uma vez que são capazes de despertar o interesse em um público mais amplo e mais jovem. São apresentados neste trabalho uma lista de 26 jogos, onde se destacam características importantes destes.

O trabalho de Pantaleão et al. (2017) descreve uma metodologia com o uso da ferramenta *Robocode* no ensino de algoritmos e programação em *Java* para alunos do Ensino Médio. A metodologia inclui a participação de discentes da graduação que cursaram a disciplina de programação. Os resultados observados mostram o interesse dos alunos do Ensino

Médio pela programação e o potencial do *Robocode* como ferramenta lúdica de apoio ao ensino, também foi possível observar o quanto a competitividade influenciou a motivação dos estudantes em aprender e superar desafios.

3.3. Discussão

Os artigos da subseção 3.2 foram importantes para este trabalho, contribuindo com o entendimento do que já foi aplicado em jogos digitais voltados ao ensino de programação, levando em conta em qual nível de escolaridade aconteceu essa aplicação. Através deles investigamos a eficácia e resultados que obtiveram, erros e acertos e também a metodologia que cada autor utilizou.

Dentre os trabalhos correlatos não é possível ver 100% de semelhança com o Carpilator, desta forma foram importante para avaliarmos como cada autor realizou a estrutura do trabalho, metodologias e resultados que obtiveram.

3.4. Tecnologias para Desenvolvimento de Jogos Digitais

Nesta subseção apresentará as ferramentas que serão utilizadas no desenvolvimento do jogo digital proposto pelo trabalho. Todas as tecnologias possuem licenças gratuitas ou versões para estudantes.

3.4.1. *Unity*

Criada pela *Unity Technologies*, a plataforma *Unity* funciona como um motor de jogos digitais que trabalha com a linguagem de programação *C#*. Além de oferecer um ambiente para programadores experientes, também oferece recursos prontos que auxiliam os desenvolvedores a focarem mais no funcionamento do seu jogo (da Silva et al., 2016).

A *Unity* é uma das mais populares *engines* para jogos, uma vez que conta com mais de 770 milhões de pessoas que jogam os títulos criados na plataforma, entre os 1000 dos maiores títulos de jogos para plataformas móveis grátis. A *Unity* é responsável por 34% de 1000 dos maiores títulos de jogos para plataformas móveis grátis, sendo escolhida principalmente pela sua interface simples e completa. No site da plataforma é possível encontrar sua documentação completa e vários tutoriais que auxiliam na aprendizagem do funcionamento da *Unity* (da Silva et al., 2016). É nesta plataforma que o jogo foi desenvolvido.

3.4.2. C#

O C# foi desenvolvido pela *Microsoft* e é uma linguagem de programação orientada a objetos que permite que os desenvolvedores criem vários tipos de aplicativos robustos e seguros que são executados no .NET (Wagner et al., 2022). Segundo Overflow (2021) é a quinta linguagem de programação mais utilizada e alguns de seus recursos são:

- Coletor de lixo: recupera de forma automática a memória ocupada por objetos não utilizados e inacessíveis;

- Tipos anuláveis: é uma proteção contra variáveis que não se referem a objetos alocados;
- Manipulação de exceção: oferece uma abordagem estruturada e extensível para detecção e recuperação de erros;
- Expressões lambda: fornecem suporte a técnicas de programação funcional;
- Linguagem de consulta integrada (LINQ): padrão criado para permitir trabalhar com dados de qualquer fonte.

Neste artigo será usado no *Unity* por ser a linguagem de programação que a plataforma utiliza.

3.4.3. *Adobe Photoshop*

É um *software* que cria ou aprimora fotografias, imagens, vídeos, animações e ilustrações 3D. Essas funcionalidades tornaram-o uma das plataformas mais usadas em computadores e de maior destaque no mundo da fotografia e do design gráfico (De Andrade, 2019).

Pra este artigo, o *Adobe Photoshop* foi utilizado para criação da interface gráfica para interação com os usuários, através da criação das telas do jogo digital, menu e botões.

3.4.4. *GitHub*

É uma plataforma de desenvolvimento colaborativo para projetos que utilizam o sistema de controle de versão *Git* e todo código armazenado pode ser aberto ou restrito, permitindo que usuários contribuam em projetos privados ou *Open Source* de qualquer lugar do mundo. Tais funcionalidades, podem levar a uma melhor tomada de decisões, redução de conflitos e aumento da produtividade (Portugal e do Prado Leite, 2015).

O *GitHub* foi usado como repositório para esse projeto, sendo utilizado, principalmente, pelos seus recursos de edição simultânea, versionamento e histórico de alteração.

3.5. *Trello*

O *Trello* é um sistema de gerenciamento de projeto *online* e funciona como um *Kanban*, organizando as atividades em cartões e filas. Suas funcionalidades intuitivas possibilitam que qualquer time configure e personalize rapidamente os fluxos de trabalho de praticamente qualquer atividade (Trello, 2022).

Neste artigo, foi usado para controle das tarefas de desenvolvimento do jogo digital.

4. Desenvolvimento

A seguinte seção contempla os passos realizados para a construção do jogo digital, apresentando a elaboração do *Game Design Document* (GDD), mostrando o questionário que foi aplicado com a turma que fez a disciplina de programação no IFSC Câmpus Lages no semestre 2022/1 e que foi usado para complementar o processo de levantamento de requisitos do jogo digital. Também apresenta o desenvolvimento do novo interpretador do jogo digital, lista de

requisitos, ordem dos requisitos implementados, controle de versões e, por fim a programação na plataforma *Unity* com as descrições da arquitetura, movimentações do personagem, desenvolvimento das estradas e o teste de usabilidade do jogo.

4.1. GDD

O Game Design Documet (GDD) apresenta toda a documentação relacionada ao design do jogo digital, dessa forma, esta subseção é composta pelo *Gameplay*, descrição dos personagens, controles e câmera do jogo digital.

4.1.1. *Gameplay*

A *gameplay* consiste em 2 modos de jogos diferentes, o primeiro é o *user-only*, onde o usuário deve apenas descobrir quais *inputs* precisa ser dado ao programa para que chegue ao objetivo final e o *code*, onde o usuário insere um código que gere uma cena idêntica à sugerida pelo nível.

4.1.2. Personagens

- Carro: representa o processador executando o código do usuário;
- Estruturas do ambiente (estruturas da programação):
  - Estrutura condicional (*If*): representado por uma bifurcação na estrada;
  - Estrutura de repetição (*While*): representado por uma rotatória;
  - Estrutura de leitura (*Read*): representado por um acostamento onde o carro deve parar para o usuário inserir dados;
  - Estrutura de início;
  - Estrutura de saída;
  - Estrutura de final;
  - *Print*: representado por uma placa com informações.

4.1.3. Controles

A *gameplay* é baseada em poucas *inputs* do jogador, sendo elas:

- Botão *Play*/*Pause*: Continua ou pausa a movimentação do carrinho (execução do código);
- Botão *Reset*: Para reiniciar o programa;
- Caixa de Texto para *Input* do código: Onde o usuário digitará o código para ser executado ou para selecionar um nível;

4.1.4. Câmera

A câmera escolhida para o jogo possui o formato *Top Down*, isto é, de cima para baixo. Assim, ao aproximá-la o jogador pode ter mais clareza dos textos escritos e da organização das estruturas.

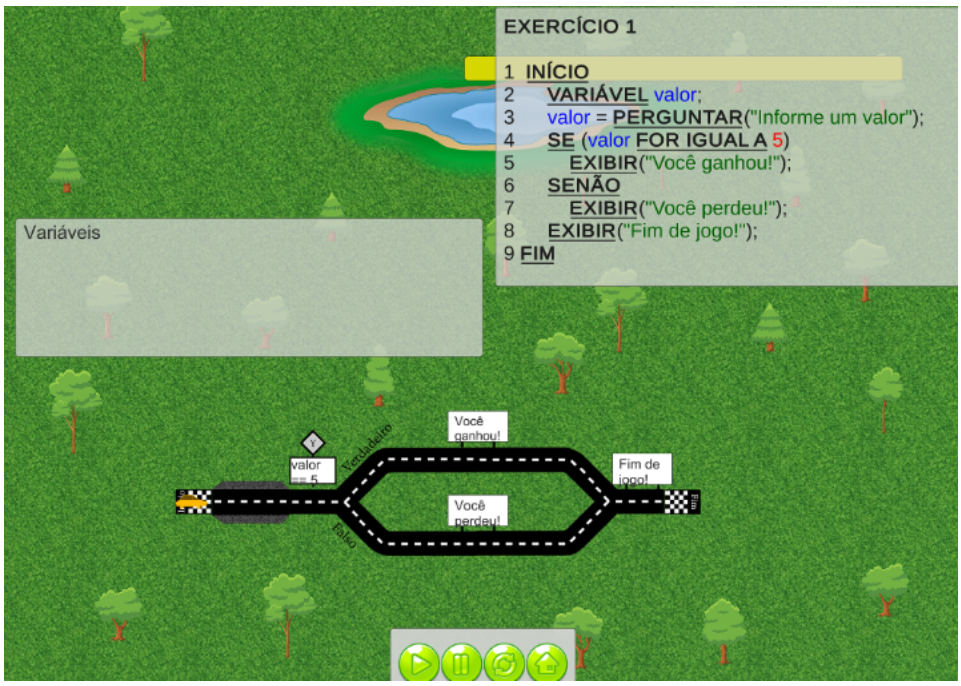

Figura 2. Imagem da câmera do jogo

4.2. Questionário aplicado para identificar problemas de aprendizado em disciplinas introdutórias de programação

Foi aplicado um questionário com a turma do semestre 2022/1 do IFSC Câmpus Lages, com o objetivo de identificar as principais dificuldades nos conteúdos que a turma estudou durante o semestre e averiguar se os alunos consideram alguma outra forma de ensino-aprendizagem diferente da que é aplicada.

O resultados da figura 3 mostra as dificuldades de cada aluno nos temas que são abordados na disciplina de programação, usando o seguinte critério de grau de dificuldade na votação: 1 - muito fácil; 2 - fácil; 3 - mais ou menos; 4 - difícil e 5 - muito difícil.

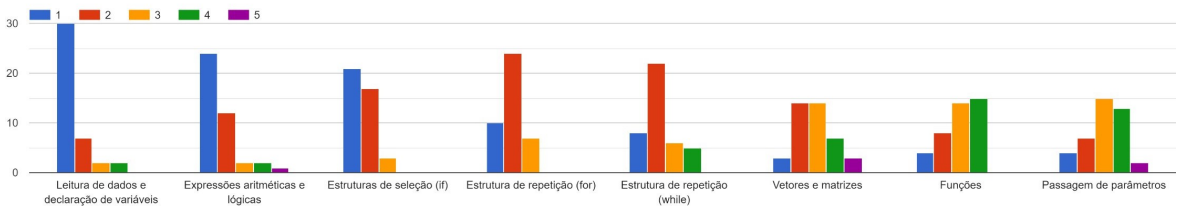

Figura 3. Marque o grau de dificuldade que teve em cada tópico da disciplina descrito abaixo. Sendo 1 - muito fácil; 2 - fácil; 3 - mais ou menos; 4 - difícil e 5 - muito difícil.

Pensando em trazer como sugestão a instituição e verificar se o nosso projeto despertaria o interesse dos discentes. Questionamos se existiria algum outro recurso de ensino-

aprendizagem que eles considerariam eficaz na disciplina. A figura 4 nos mostra os resultados mais votados pelos alunos, podendo observar que a inclusão dos jogos digitais durante as aulas está em segundo lugar com 43,9% como uma ferramenta válida para melhorar a aprendizagem.

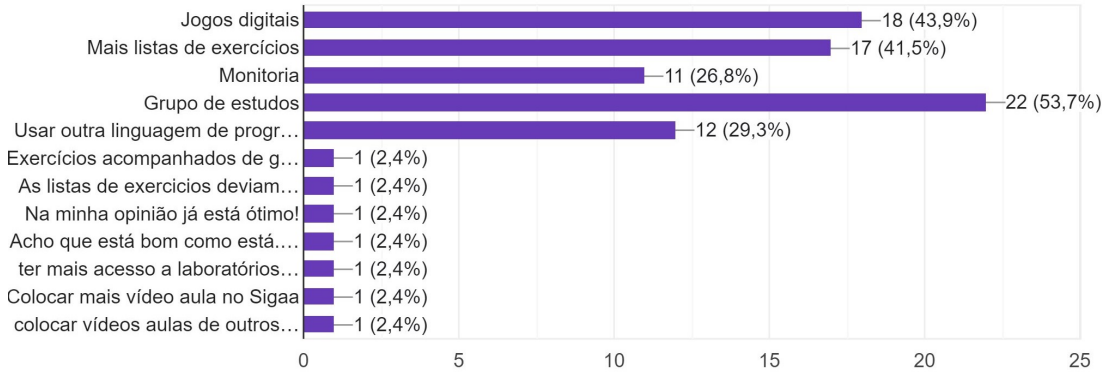

Figura 4. Qual estratégia facilitaria a aprendizagem da disciplina?

Portanto, foi possível verificar que os alunos temem principalmente, a parte lógica e de passagem de parâmetros e observar que os alunos consideram interessante a ideia de um complemento com ferramentas lúdicas na hora do ensino-aprendizagem da disciplina. Apesar da detecção que passagem de parâmetros é um dos assuntos que os discentes mais tem dificuldades, ela não foi implementada pois o foco do jogo digital nesse primeiro momento é nas funções básicas.

4.3. Lista de Requisitos do Interpretador

Lista de requisitos para o Interpretador.

- RF01 - Realizar a análise léxica. Analisador léxico é responsável por ler o código digitado pelo usuário e reconhecer os *tokens* da linguagem;
- RF02 - Realizar a análise sintática. Analisador sintático é responsável por transformar esses *tokens* em uma árvore sintática abstrata (AST) que possa ser lida pelo Executor;
- RF03 - Gerar Cenas de maneira procedural. O processo que trabalhará com o *Unity* para gerar a cena condizente ao código digitado pelo usuário a partir da AST;
- RF04 - Interpretar o código em tempo de execução. O processo que recebe a AST e é responsável por traduzir o código para C# em tempo de execução do jogo e retornar os devidos comandos ao jogo *Unity*.

4.4. Desenvolvimento do interpretador que será utilizado no jogo digital

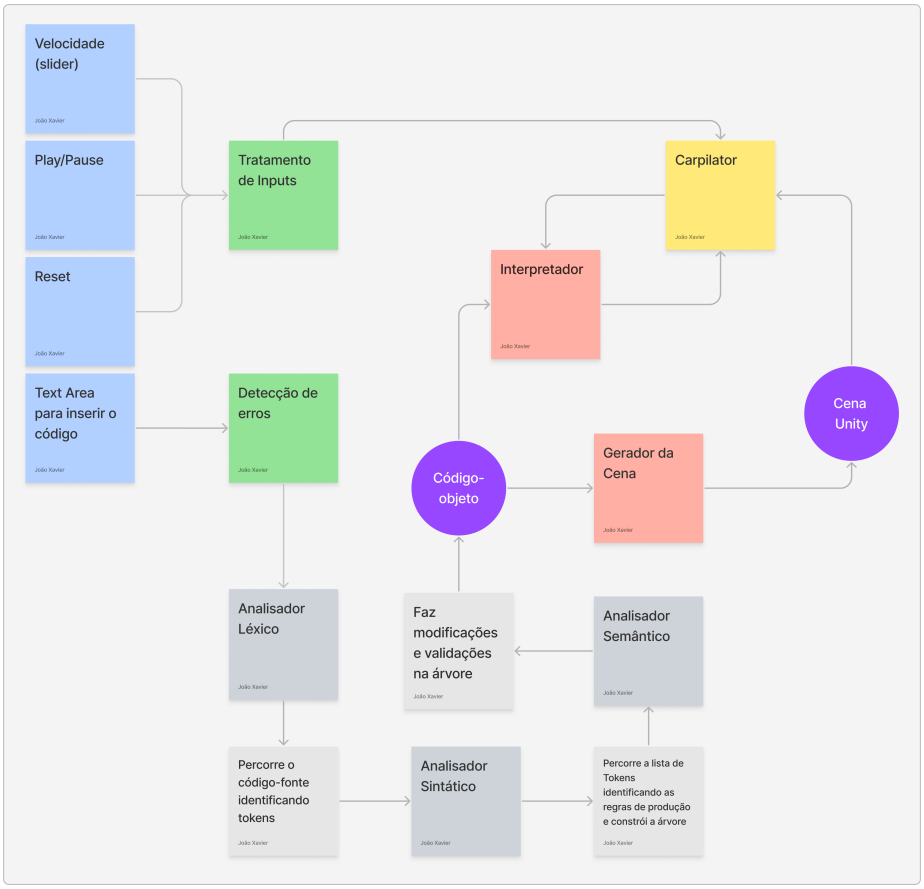

Figura 5. Fluxograma do Sistema

A Figura 5 mostra um fluxograma geral do sistema, onde em azul temos as *inputs* do usuário, em verde os módulos que fazem o tratamento das *inputs* para alterar o comportamento do jogo, em cinza as etapas do ”Compilador”e em roxo os objetos de saída do programa.

A implementação iniciou-se pelo interpretador da linguagem de programação do jogo, a C#. Dividido em partes:

- Analisador Léxico;
  Essa etapa recebe como *input* o próprio código-fonte da linguagem em forma de texto, e vai transformar esse código em uma lista de *tokens* através de um Autômato finito determinístico, que vai gerar tais *tokens* buscando dentro da *string* os padrões definidos da linguagem.
- Analisador Sintático;
  Essa etapa recebe como *input* a lista de *tokens* da etapa anterior, e é responsável por agrupar os *tokens* em uma árvore sintática abstrata através das regras gramaticais definidas para formar os construtos da linguagem

- Gerador de Cena;
  Essa etapa recebe a árvore sintática abstrata e é responsável por gerar a Cena 3D no *Unity* com todas as sub-cenas e componentes do game.
- Executor.
  Por fim o executor é o responsável por dar vida á cena, receber as inputs do usuário passar para o interpretador, controlar o código de acordo com elas e mostrar toda esse comportamento para o usuário graficamente.

A Figura 6 mostra o método responsável por reconhecer as gramáticas da linguagem com base na lista de *tokens* que foi obtida pela etapa da análise léxica.

```
private Statement Statement()
{
    return Current.TokenType switch
    {
        TokenType.Identifier => Identifier(),
        TokenType.ReservedWord => ReserverdWord(),
        _ => throw new UnexpectedToken(Current, TokenType.Identifier, TokenType.ReservedWord),
    };
}
```

Figura 6. Função para reconhecimento sintático dos comandos do programa

A Figura 7 mostra o método responsável por reconhecer as possíveis gramáticas ao analisar um identificador.

```
private Statement Identifier()
{
    var identifier = (Identifier)Assert(TokenType.Identifier);

    return Current.TokenType switch
    {
        TokenType.ParenthesisOpen => FunctionCall(identifier),
        TokenType.Attribution => AssignmentExpression(identifier),
        TokenType.Identifier => VariableDeclaration(identifier),
        _ => throw new UnexpectedToken(identifier, TokenType.Identifier, TokenType.ParenthesisOpen, TokenType.Attribution),
    };
}
```

Figura 7. Função para reconhecimento sintático de identificadores

A Figura 8 mostra o método responsável por reconhecer a gramática de uma chamada de função.

```
private FunctionCall FunctionCall(Identifier identifier)
{
    Assert(TokenType.ParenthesisOpen);

    var parameters = new List<IValuable>();

    DoUntil(TokenType.ParenthesisClose, () =>
    {
        parameters.Add(GetExpression());
        AssertOptional(TokenType.Comma);
    });

    var functionCall = new FunctionCall(identifier, parameters);

    Assert(TokenType.ParenthesisClose);
    AssertOptional(TokenType.Semicolon);

    return functionCall;
}
```

Figura 8. Função para reconhecimento sintático de uma chamada de função

A Figura 9 mostra o método responsável por reconhecer a gramática de uma expressão.

```
private IValuable GetExpression()
{
    var leftValue = Assert
    (
        TokenType.StringValue,
        TokenType.IntValue,
        TokenType.FloatValue,
        TokenType.BoolValue,
        TokenType.Identifier
    );

    if (leftValue.TokenType == TokenType.Identifier && Current.TokenType == TokenType.ParenthesisOpen)
    {
        return FunctionCall((Identifier)leftValue);
    }

    if (IsEOL())
    {
        return (IValuable)leftValue;
    }

    return new BinaryExpression((IValuable)leftValue, GetOperator(), GetExpression());
}
```

Figura 9. Função para reconhecimento sintático de expressões

4.5. Desenvolvimento do jogo digital no *Unity*

Nas seguintes seções serão apresentadas as classes desenvolvidas para implementar o jogo na *Unity*, essa implementação tem uma arquitetura baseada em componentes, ou seja, baseia-se em componentes independentes, substituíveis e modulares, as quais gerenciam a complexidade e motivam a reutilização.

4.5.1. Arquitetura do jogo digital

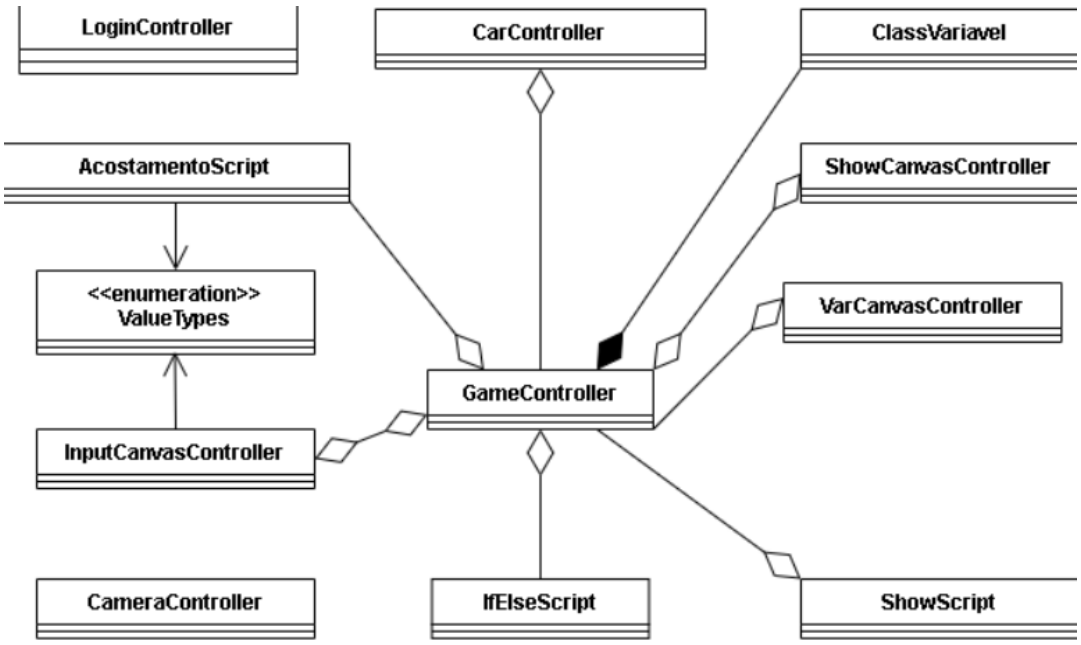

Figura 10. Arquitetura do jogo digital

4.5.2. *GameObject* Acostamento

Este *GameObject* tem como objetivo manipular o carro e fazer que quando ele passar por cima do *Collider* apareça na tela a caixa de *input*. Dentro do *GameObject* Acostamento temos os seguintes componentes: *Transform*, *BoxCollider* e o *script* Acostamento, como podemos observar na figura 11 e na 12.

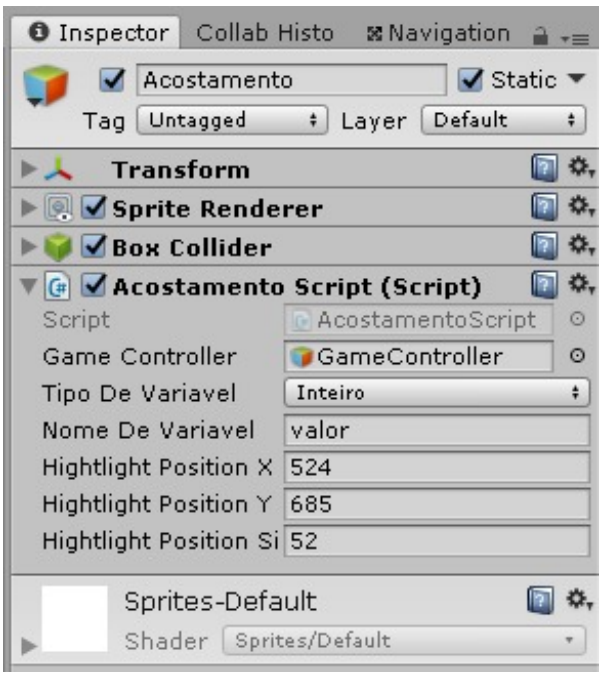

Figura 11. GameObject Acostamento

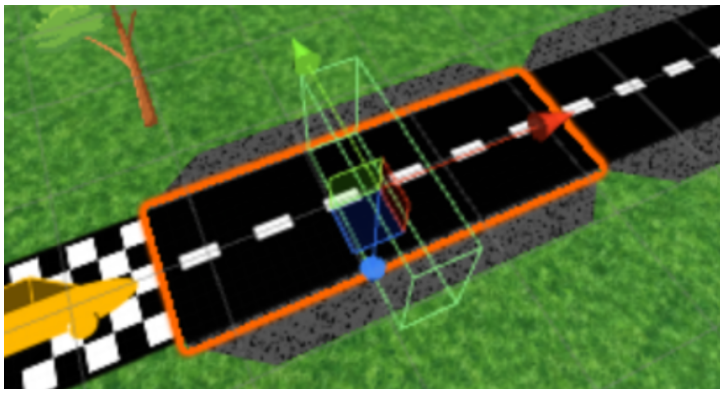

Figura 12. Parte do jogo que representa o GameObject Acostamento

Conforme, é possível ver na figura 13, o atributo da linha 10 é responsável por fazer a comunicação com o *GameObject Controller*, o da linha 11 e 12 definem qual é o nome e o tipo da variável que será lida pelo *GameObject InputCanvas*. Os três atributos das linhas 14,15,16 definem o X, Y e Z do quadrado amarelo que faz o *Highlight* do código. Na linha 19 o atributo segura a referência ao componente *Collider* do *GameObject* e na linha 21 o atributo faz referência ao componente *GameController*, que faz a comunicação entre todos os objetos da cena.

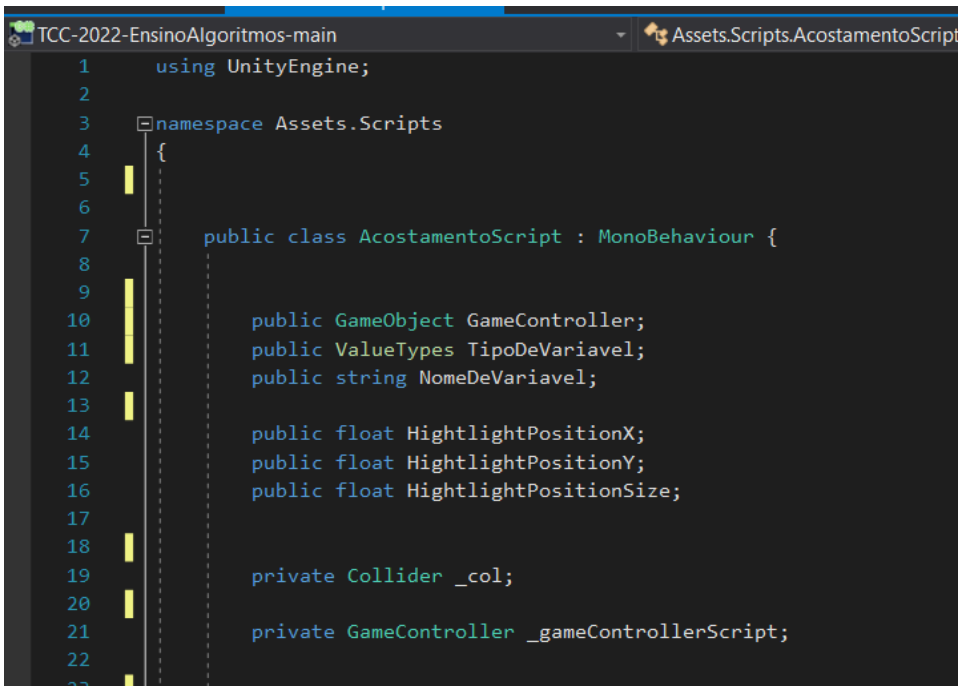

Figura 13. GameObject Acostamento - Parte 1 do código

No método *Start*, como podemos observar na figura 14 ao dar o *play* no *Unity*, o *script* pega a referência do componente *Collider* definido dentro do *GameObject* Acostamento e coloca no atributo. Além disso, pega a referência do componente do *script GameController* de dentro do Objeto *GameController*.

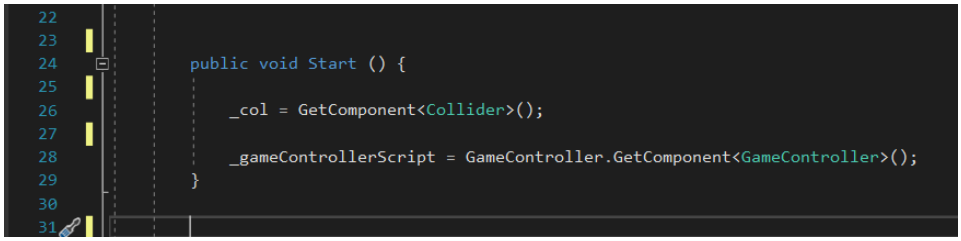

Figura 14. GameObject Acostament - Parte 2 do código

O método *OnTriggerEnter* é chamado sempre que ocorre uma colisão entre *Colliders* de diferentes objetos, sendo chamado toda vez que um outro objeto invade um *Collider* definido como *Trigger*. A diferença entre *Collider* e e o *Trigger* é uma área de ativação e o *Collider* impede que outros objetos invadam essa região. Nas linhas 35 e 36 da figura 15, o *_gameControllerScript* configura o *GameObject TextHighliter* que mostra em qual ponto do código o carrinho está e o *GameController* é um objeto centralizador que fará a comunicação entre os objetos, todo este processo é possível observar na figura 15.

```
32      public void OnTriggerEnter(Collider other)
33      {
34
35          _gameControllerScript.SetHightlightTextColor(255f,255f,0f);
36          _gameControllerScript.MoveHightlightToPosition(HightlightPositionX, HightlightPositionY, HightlightPositionSize);
37
38
39          _gameControllerScript.StopCar();
40
41
42          _gameControllerScript.ActivateInputCanvas(NomeDeVariavel, TipoDeVariavel);
43          _col.enabled = false;
44      }
45  }
46  }
47
```

Figura 15. GameObject Acostamento - Parte 3 do código

4.5.3. *GameObject* câmera

A câmera fica posicionada em cima da cena e apontando para baixo. Dentro da câmera, o *script CâmeraController* controla tanto o posicionamento do *GameObject Camera*, como o zoom na tela. Na figura 16 o eixo X é representado pelo vetor vermelho e o eixo Z é representado pelo vetor azul.

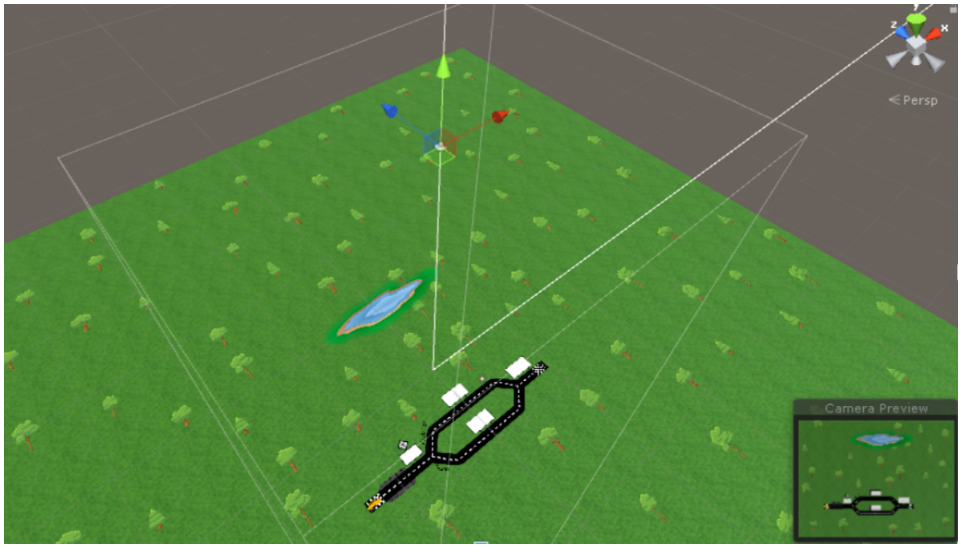

Figura 16. Parte do jogo que mostra a Script Câmera

Na figura 17, na linha 8 vemos o atributo que fará referência a câmera principal na cena, já as linhas 11 e 12 são atributos que controlam a velocidade que a tela movimenta-se quando modificamos o X e Z da câmera, tais atributos são necessários pois quando a câmera está com muito zoom, ela deve movimentar-se mais lentamente. Nas linhas 15,16,17 e 18 são atributos que controlam o máximo X e Z que a câmera pode andar, para que o jogador não vá com a câmera em locais onde não existe mais cenário, sendo atributos públicos para que o programador possa alterar esses valores pelo editor do *Unity* e os valores definidos no

código são os valores que são atribuídos inicialmente quando o *script* é vinculado, conforme é possível verificar na figura 18.

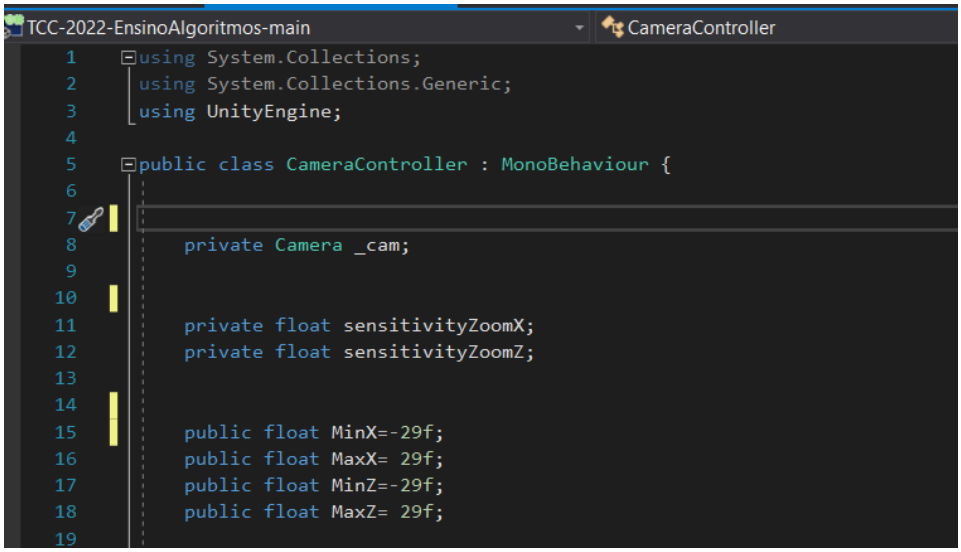

Figura 17. Script Câmera - Parte 1 do código

Figura 18. Script Câmera

Na linha 26 na figura 19, quando apertamos o *play* no *Unity*, a primeira coisa que esse componente *script* faz é ligar a câmera na cena e esse atributo no código, desta forma, podemos manipular seu comportamento via código.

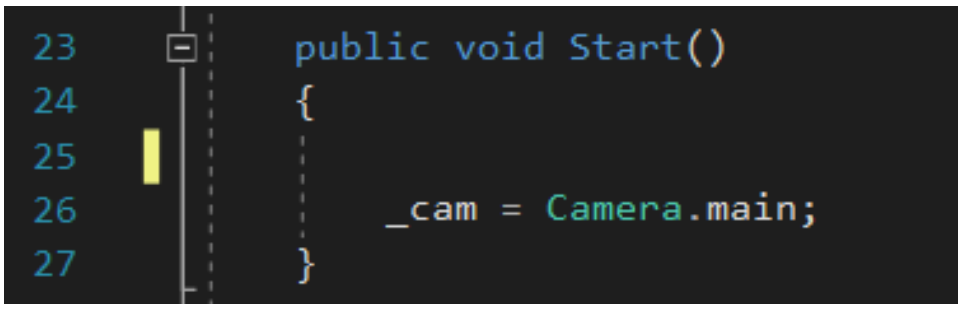

Figura 19. Script Câmera - Parte 2 do código

No método *Update*, como vemos na figura 20 na linha 33 e 34 a instrução *_cam.orthographicSize* controla o zoom atual da câmera, quanto maior, mais longe. O código das linhas 33 e 34 através do zoom da câmera determinam a velocidade em que a câmera poderá mover-se, pois quanto mais perto, a câmera deve mover-se mais lentamente conforme fazer o movimento com o mouse. Na linha 37, se o usuário mexer no *scroll* do mouse para cima, deve aumentar o zoom na tela, na linha 41, se o usuário mexer no *scroll* do mouse para baixo, deve diminuir o zoom na tela e a linha 45 é acionada caso o usuário aperte o botão esquerdo (0) ou pressionar o *scroll* do mouse (2), ao pressionar esses botões as linhas 48 e 52 fazem o movimento de *padding* na tela.

```
30  public void Update()
31  {
32
33      sensitivityZoomX = 0.05f * _cam.orthographicSize;
34      sensitivityZoomZ = 0.05f * _cam.orthographicSize;
35
36
37      if (Input.GetAxis("Mouse ScrollWheel") > 0 && _cam.orthographicSize > 5.0f)
38          _cam.orthographicSize -= .5f;
39
40
41      if (Input.GetAxis("Mouse ScrollWheel") < 0 && _cam.orthographicSize < 25.0f)
42          _cam.orthographicSize += .5f;
43
44
45      if (Input.GetMouseButton(0) || Input.GetMouseButton(2))
46      {
47
48          if ((_cam.transform.position - _cam.transform.right* Input.GetAxis("Mouse X") * sensitivityZoomX).x >
49              MinX && (_cam.transform.position - _cam.transform.right * Input.GetAxis("Mouse X") * sensitivityZoomX).x < MaxX)
50              _cam.transform.position -= _cam.transform.right * Input.GetAxis("Mouse X") * sensitivityZoomX;
51          if ((_cam.transform.localPosition - _cam.transform.up * Input.GetAxis("Mouse Y") * sensitivityZoomZ).z >
52              MinZ && (_cam.transform.localPosition - _cam.transform.up * Input.GetAxis("Mouse Y") * sensitivityZoomZ).z < MaxZ)
53              _cam.transform.localPosition -= _cam.transform.up * Input.GetAxis("Mouse Y") * sensitivityZoomZ;
54      }
55  }
56  }
```

Figura 20. Script Câmera - Parte 3 do código

4.5.4. *GameObject CarController*

Nesta parte, conforme conseguimos visualizar na imagem 21, dento do *GameObject Car* estão presentes os componentes *script CarController*, *Collider* e o *NavMeshAgent*, o qual trabalha com a inteligencia artificial dos *GameObjects* para que possam andar em um terreno, ele só

consegue andar por cima de objetos que se movimentam, esse atributo fica visível na imagem 22.

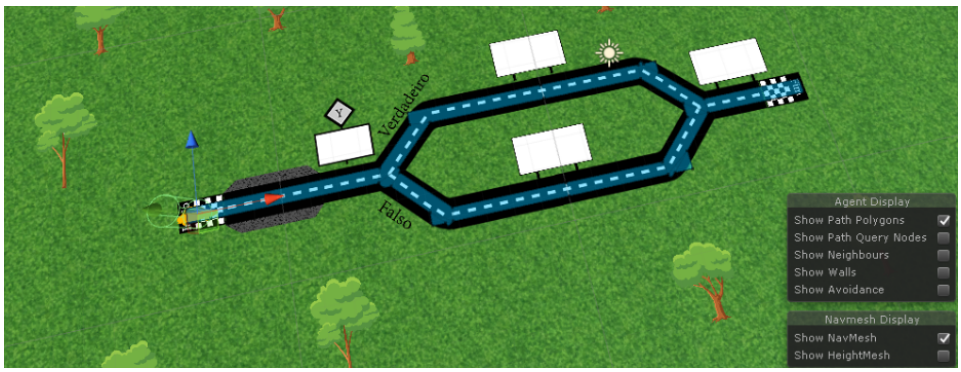

Figura 21. Parte do jogo que mostra o Script CarController

No método *Start*, tem seu início na linha 14 conforme a figura 23, o *GameObject CarController* é localizado dentro do *GameObject Car* e é ligado com o *script* e tenta direcioná-lo para o destino definido. Já o método *public void SetNavMeshAgentSpeed (float speed)* que começa na linha 22, define a velocidade atual do carro, sendo possível modificar um atributo dentro do componente *NavMeshAgent* do carro.

```
12
13
14          public void Start () {
15
16              _navMeshAgent = GetComponent<NavMeshAgent>();
17
18              _navMeshAgent.SetDestination(Destination);
19          }
20
21
22          public void SetNavMeshAgentSpeed (float speed)
23          {
24
25              _navMeshAgent.speed = speed;
26          }
27      }
28  }
29
```

Figura 23. Script CarController - parte 2 do código

4.5.5. *ClassVariavel*

Conforme a figura 24, a classe é usada para criar a lista de variáveis usadas no *GameObject* e no *GameController*.

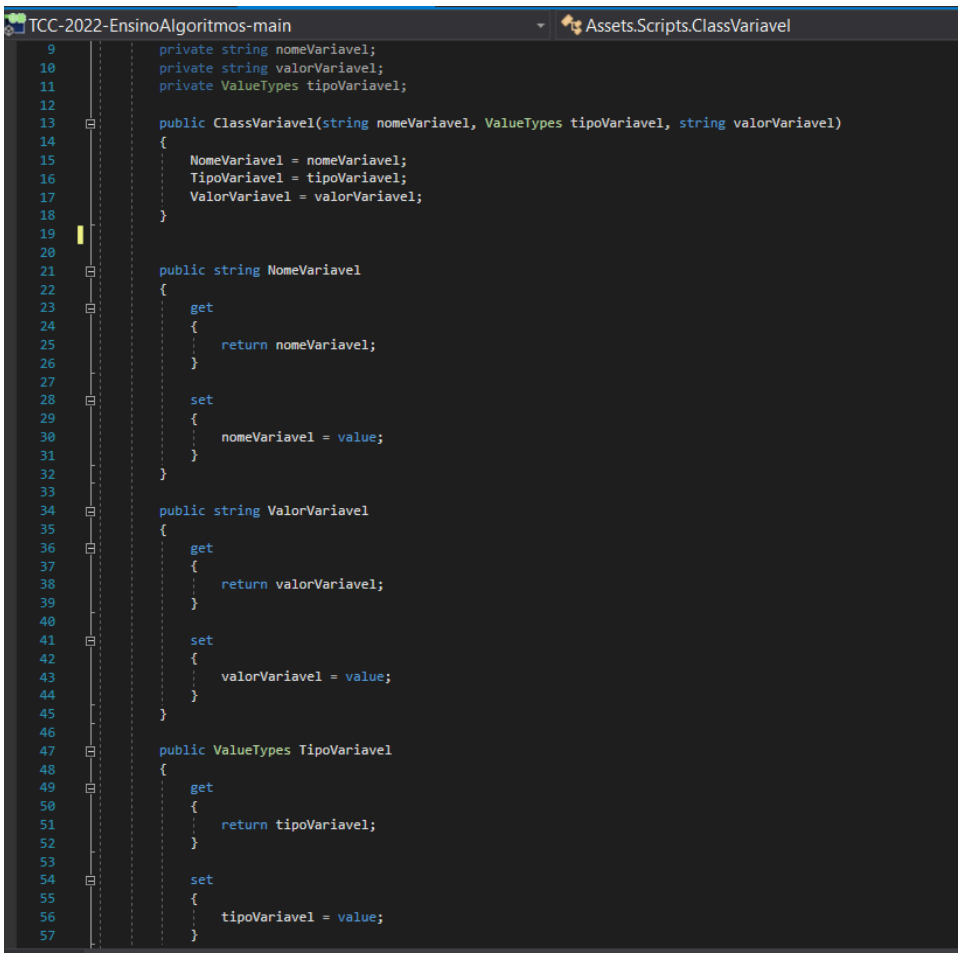

Figura 24. Class Variável

4.5.6. *EnumTypes*

Conforme observamos na figura 25, o *EnumTypes* é um tipo de variável onde definimos algum valores para ele, no caso inteiro, real e texto

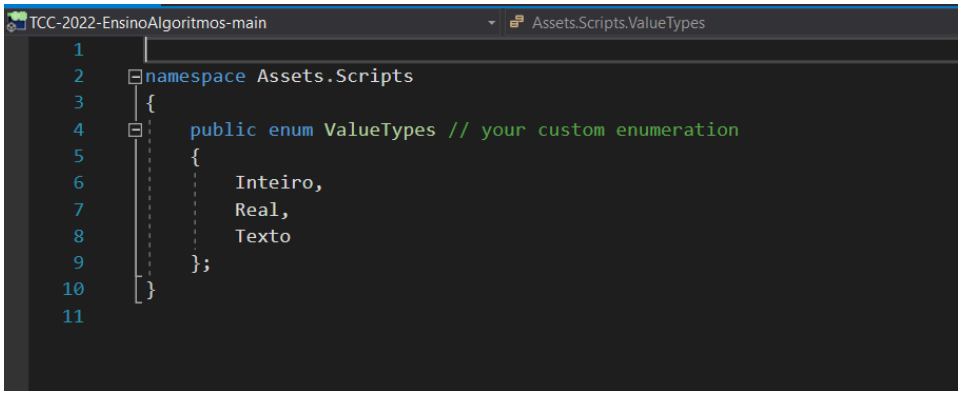

Figura 25. EnumTypes

4.5.7. *GameController*

O *GameController* é a classe centralizadora do jogo digital, a qual possui todas as interações entre os objetos, ou seja, tudo que tem ação ou reação passa por esta classe. Na figura 26, notamos que todos os *GameObjects* que o *GameController* pode manipular estão públicos para que possa ser feito a vinculação manual dos *GameObjects* pelo *Unity*.

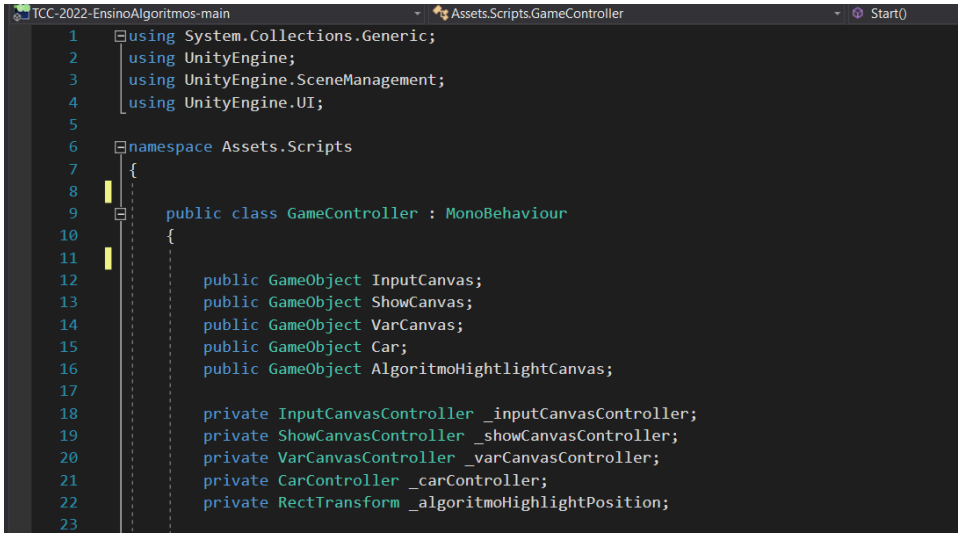

Figura 26. Game Controller - Parte 1 do código

Já na figura 27, na linha 26 temos a lista que armazena todas as variáveis adicionadas pelo código. Da linha 28 a 36 temos o método *Start*, o qual pega a referência do *InputCanvasController* que está dentro do *GameObject InputCanvas* e através dessas *scripts* o *GameController* pode enviar mensagens para esses objeto. Na linha 35 deste método é pego a referência do *RectTransform* para futuramente manipular as coordenadas do *TextHighlighter*.

Figura 27. Game Controller - Parte 2 do código

Na linha 48 da figura 28, temos o método que modifica a cor do *TextHightlighter*, já na linha 54 a 62 o método pega o valor de uma variável que exista dentro da *List* ListaDeVariaveis e retorna o conteúdo encontrado.

```
47
48      public void SetHightlightTextColor(float r, float g, float b)
49      {
50          _algoritmoHighlightPosition.GetComponent<Image>().color = new Color(r, g, b);
51      }
52
53
54      public string GetValorDeVariavel(string nomeVariavel)
55      {
56          foreach (var variavel in ListaDeVariaveis)
57          {
58              if (nomeVariavel == variavel.NomeVariavel)
59                  return variavel.ValorVariavel;
60          }
61          return "";
62      }
```

Figura 28. Game Controller - Parte 3 do código

Na figura 29, a partir da linha 66 observamos o método que é chamado toda vez que o usuário faz uma entrada válida no *InputCanvas*, sendo adicionada uma nova variável e imprimindo no *canvas*de variáveis de texto.

```
64
65
66      public void AddVariavel(string nomeDeVariavel, ValueTypes tipoDeVariavel, string valorDeVariavel)
67      {
68
69          ListaDeVariaveis.Add(new ClassVariavel(nomeDeVariavel, tipoDeVariavel, valorDeVariavel));
70
71          _varCanvasController.AddVarText(nomeDeVariavel + ": " + valorDeVariavel + "\n");
72      }
73
```

Figura 29. Game Controller - Parte 4 do código

O método que faz aparecer o *InputCanvas* na tela pode ser observado na figura 30 e o método que faz aparecer o *ShowCanvas* na tela está representado na imagem 31.

```
74
75      public void ActivateInputCanvas(string nomeDeVariavel, ValueTypes tipoDeVariavel)
76      {
77          _inputCanvasController.ActivateInputCanvas(nomeDeVariavel, tipoDeVariavel);
78      }
```

Figura 30. Game Controller - Parte 5 do código

```
80
81      public void ActivateShowCanvas(string text)
82      {
83          _showCanvasController.ActivateShowCanvas(text);
84      }
```

Figura 31. Game Controller - Parte 6 do código

Na figura 32 temos o método chamado toda vez que precisamos parar o carro e na imagem 33 observamos o método chamado toda vez que precisamos fazer com que o carro volte a andar.

```
86
87          public void StopCar()
88          {
89              _carController.SetNavMeshAgentSpeed(0);
90          }
```

Figura 32. Game Controller - Parte 7 do código

```
92
93          public void ResumeCar()
94          {
95              _carController.SetNavMeshAgentSpeed(3f);
96          }
97
```

Figura 33. Game Controller - Parte 8 do código

Na figura 34 na linha 99 a 102 temos representado o método chamado toda vez que precisamos recarregar a cena para seu estado inicial e na linha 105 a 108 temos o método chamado toda vez que precisamos recarregar uma cena desejada.

```
97
98
99          public void ReloadScene()
100         {
101             SceneManager.LoadScene(SceneManager.GetActiveScene().name);
102         }
103
104
105         public void LoadScene(string nome)
106         {
107             SceneManager.LoadScene(nome);
108         }
109     }
110 }
111
```

Figura 34. Game Controller - Parte 9 do código

4.5.8. *InputCanvasController*

Na figura 35, a linha 10 faz referência ao *GameObject GameController*, a linha 12 referencia para o *script GameController* dentro do *GameObject GameController*, a linha 14 faz referência para o texto dentro do *GameObject InputCanvas* e na linha 16 faz referência para o *InputText* dentro do *GameObject InputCanvas*.

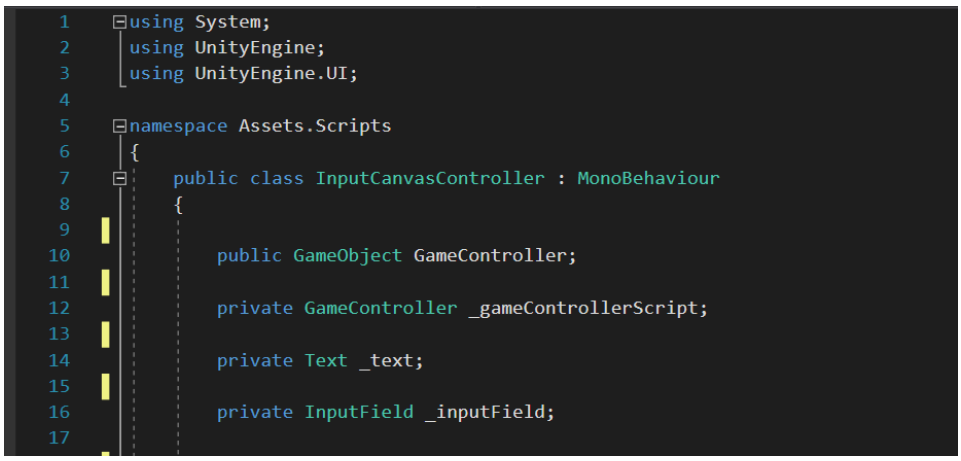

Figura 35. InputCanvas Controller - Parte 1 do código

Na imagem 36 são atributos que controlam a variável que será adicionada pelo *InputCanvas*.

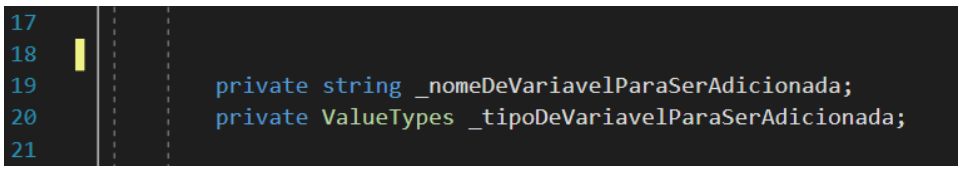

Figura 36. InputCanvas Controller - Parte 2 do código

Na figura 37 temos o método que procura dentro do *GameObject* que está vinculado esse *script*, que no caso é o *GameObject InputCanvas*, ele busca um componente chamado *InputField* e guarda a referência no atributo *_inputField*.

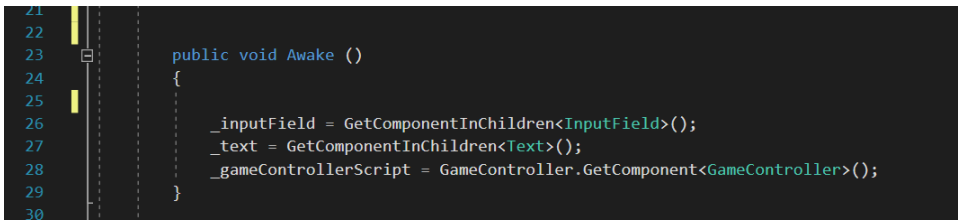

Figura 37. InputCanvas Controller - Parte 3 do código

Na figura 38 o método é chamado quando é feito o *submit* do *InputCanvas*. Na linha 38 Caso o tipo da variável não seja texto, será necessário tentar converter para ver se é um *float* ou *int* válido. Na linha 51 acontece se a entrada é válida e então a linha 57 desativa o *InputCanvas* e a 59 define se o carro pode continuar o movimento.

```
32
33      public void GetInput(string entrada)
34      {
35          _inputField.text = "";
36          _inputField.ActivateInputField();
37
38          if (!_tipoDeVariavelParaSerAdicionada.Equals("Texto"))
39          {
40              try
41              {
42                  var unused = Single.Parse(entrada);
43              }
44              catch
45              {
46                  return;
47              }
48          }
49
50
51          if (entrada != "")
52          {
53
54              _gameControllerScript.AddVariavel(_nomeDeVariavelParaSerAdicionada, _tipoDeVariavelParaSerAdicionada,
55                  entrada);
56
57              gameObject.SetActive(false);
58
59              _gameControllerScript.ResumeCar();
60          }
```

Figura 38. InputCanvas Controller - Parte 4 do código

O método representado na figura 39 ativa o *InputCanvas*, que chamamos pelo *GameObject Acostamento* e a linha 69 liga o objeto. Já a partir da linha 77 temos o método para definir o texto que aparece no InputCanvas.

```
65
66
67      public void ActivateInputCanvas(string nomeDeVariavel, ValueTypes tipoDeVariavel)
68      {
69          gameObject.SetActive(true);
70          _nomeDeVariavelParaSerAdicionada = nomeDeVariavel;
71          _tipoDeVariavelParaSerAdicionada = tipoDeVariavel;
72          SetInputFieldText("Entre com o valor da variável: [" + nomeDeVariavel + "] do tipo: [" +
73              tipoDeVariavel + "]");
74          _inputField.ActivateInputField(); // Request focus
75      }
76
77      private void SetInputFieldText(string text)
78      {
79          _text.text = text;
80      }
81
82      public void ConfirmInputCanvas()
83      {
84          GetInput(_text.text);
85      }
86
87  }
88  }
```

Figura 39. InputCanvas Controller - Parte 5 do código

4.5.9. *ShowCanvasController*

Semelhante com o *InputCanvasController*, conforme a a figura 40 mostra, na linha 10 observamos o atributo que faz referência para o *GameObject GameController*, a linha 12 referencia o botão confirmar do *ShowCanvas*, a linha 14 referencia o texto do *ShowCanvas* e a linha 16 faz referência para o Componente *Script* do *GameController*.

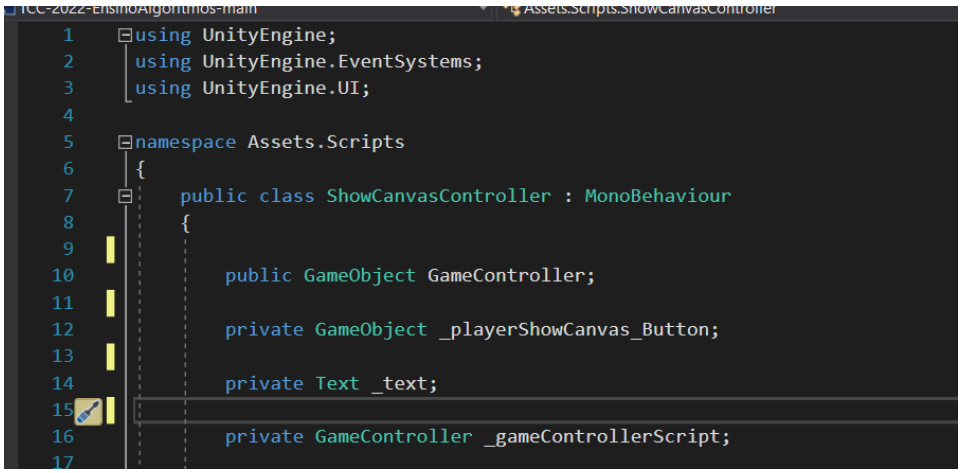

Figura 40. ShowCanvasController - Parte 1 do código

Na linha 26 da figura 41, temos o método que é chamado toda vez que precisamos mostrar o *ShowCanvas*. Já na figura 42, na linha 35 temos o método chamado para definir o texto do *ShowCanvas* e na linha 41 temos o método chamado quando o botão confirmar do *ShowCanvas* é apertado, no qual na linha 45 manda o carro parar quando ocorrer uma colisão.

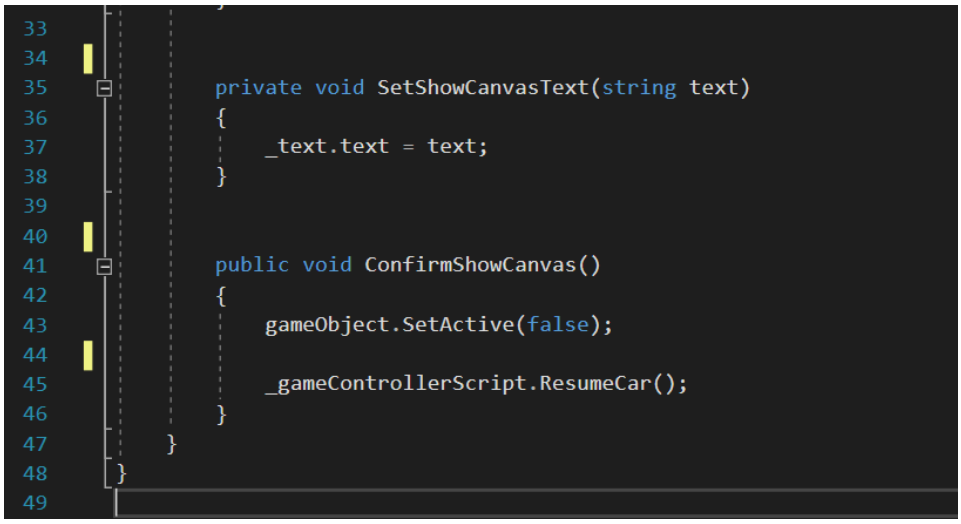

Figura 42. ShowCanvasController - Parte 3 do código

4.5.10. *ShowScript*

O *ShowScript* controla o *Collider* que aciona o *ShowCanvasController*.Na figura 43, a linha 9 controla o texto que será exibido no *GameObject Show*, a linha 11 aponta para o *Rect-Transform* de um *Canvas* que controla o tamanho da fonte do *GameObject Show* e a linha 13 aponta para o *GameObject GameController*.Já na figura 44 os códigos determinam a posição do *TextHighlither*.

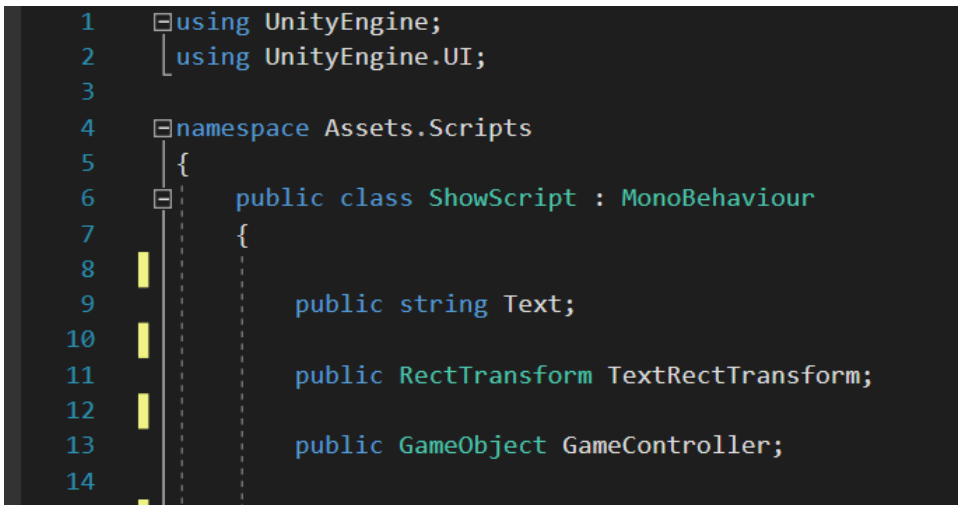

Figura 43. ShowScript - Parte 1 do código

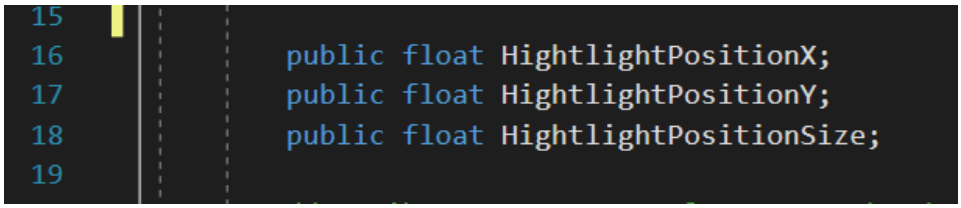

Figura 44. ShowScript - Parte 2 do código

Na figura 45 os atributos das linhas 21 e 22 controlam o tamanho da fonte com base no *zoom* e no posicionamento da câmera, a linha 25 é responsável por apontar para o *collider* dentro do *GameObject Show*, o qual o carrinho irá colidir. Já a linha 27, aponta para o componente *script* dentro do *GameObject GameController* e a linha 29 aponta para o componente *Text* dentro do *Canvas* do *GameObject Show*.

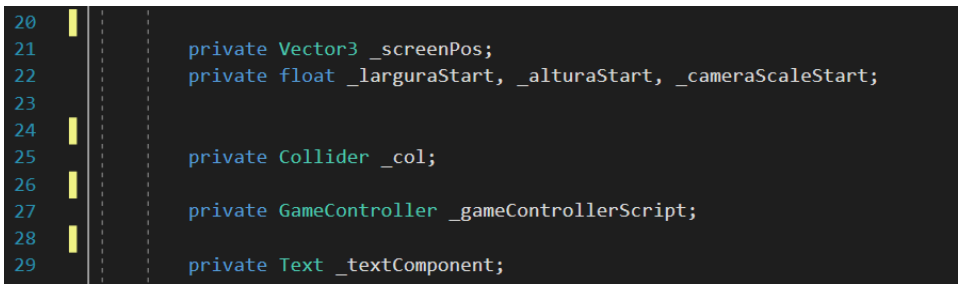

Figura 45. ShowScript - - Parte 3 do código

O método *start* está representado na figura 46, o qual pega as configurações iniciais do *GameObject Text* dentro do *Canvas* do *GameObject Show*, dessa forma na linha 34 o valor referenciado é o *Width*, na linha 35 *Height* e na linha 36 temos o zoom da câmera referenciado.

A linha 40 é a referência do *Collider* e passo para o atributo no script e na linha 42 é referenciado o componente *script* dentro do *GameObject GameController* e por fim nas linhas 45 e 46 ocorre a definição do componente *Text*.

```
30
31      public void Start()
32      {
33
34          _larguraStart = TextRectTransform.sizeDelta.x;
35          _alturaStart = TextRectTransform.sizeDelta.y;
36          _cameraScaleStart = Camera.main.orthographicSize;
37
38
39
40          _col = GetComponent<Collider>();
41
42          _gameControllerScript = GameController.GetComponent<GameController>();
43
44
45          _textComponent = GetComponentInChildren<Text>();
46          _textComponent.text = Text;
47      }
```

Figura 46. ShowScript

Na figura 47 é exibido o método *LateUpdate*, nas linhas 52 e 53 ocorre o ajuste na posição dos *canvas* de texto na tela e a linha 56 ajusta a escala dos campos *canvas* de texto. Já na figura 48 está representado o método *OnTriggerEnter*, nas linhas 65,66 e 67 ocorre o uso do *GameController* para mudar a posição do *text highlither*, a linha 68 possui o comando de parada ao carro colidir e a linha 71 desativa o *Collider*.

```
49      public void LateUpdate()
50      {
51
52          _screenPos = Camera.main.WorldToScreenPoint(transform.position);
53          TextRectTransform.position = new Vector3(_screenPos.x, _screenPos.y, 0f);
54
55
56          TextRectTransform.sizeDelta = new Vector2(_larguraStart *
57          (2 * _cameraScaleStart - Camera.main.orthographicSize) / _cameraScaleStart, _alturaStart *
58          (2 * _cameraScaleStart - Camera.main.orthographicSize) / _cameraScaleStart);
59      }
```

Figura 47. ShowScript

```
61
62      public void OnTriggerEnter(Collider other)
63      {
64
65          _gameControllerScript.SetHightlightTextColor(255f, 255f, 0f);
66          _gameControllerScript.MoveHightlightToPosition(HightlightPositionX, HightlightPositionY,
67          HightlightPositionSize);
68          _gameControllerScript.ActivateShowCanvas(Text);
69
70
71          _gameControllerScript.StopCar();
72
73          _col.enabled = false;
74      }
```

Figura 48. ShowScript

4.5.11. *VarCanvasController*

Esse *script* controla o painel de variáveis, seu código está representado na 49. Na linha 12 começa o método *start*, o qual na linha 15 encontra-se a primeira referência de um componente *Text* em seus filhos.

4.5.12. *IfElseScript*

Conforme a figura 50, observamos que na linha 17 o texto dentro do objeto *If/Else* será a condição, o atributo da linha 20 aponta para o *canvas* que exibe o texto do *GameObject If/Else*, usado para ajustar o texto conforme o zoom da tela acontece, na linha 22 o atributo aponta para o *GameObject GameController*.Já nas linhas 25 e 26 são os *Gameobjects* que apontam para o local onde o fluxo da rodovia será cortado. As linhas 29,30,31,34,35 e 36 determinam a posição do *TextHighlither*

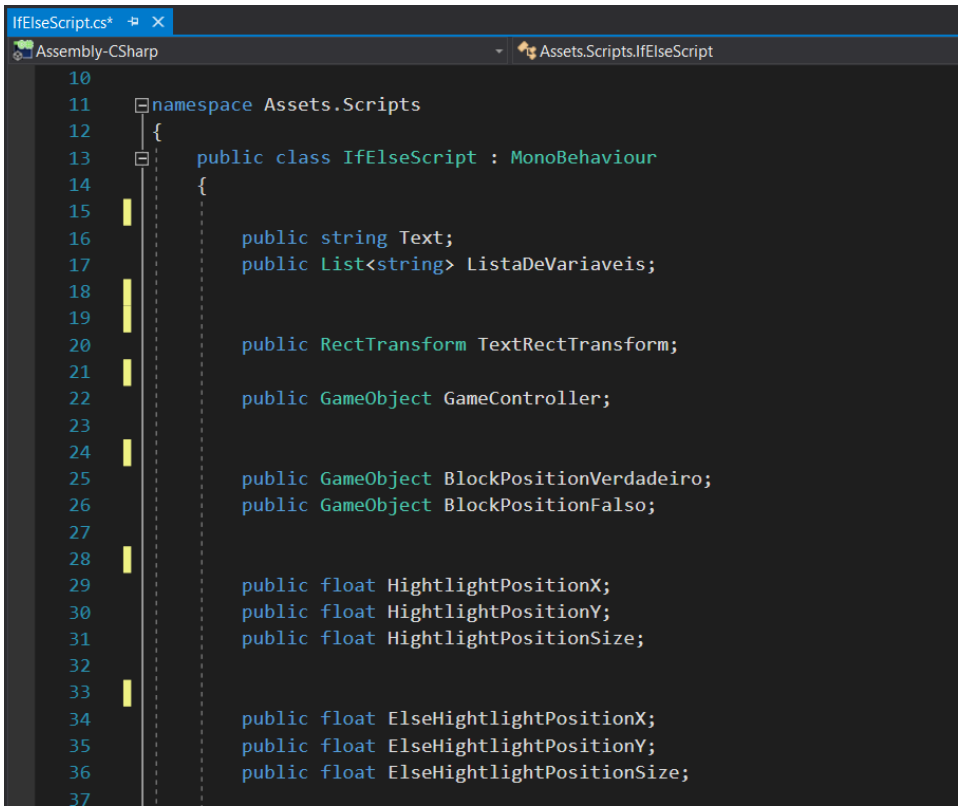

Figura 50. IfElseScript - Parte 1 do código

Na figura 51, na linha 39 o atributo guardar a referência do componente *script GameController*, nas linhas 43,44 e 45 temos os atributos que verificam o posicionamento da câmera para ajustar o tamanho do texto exibido dentro do *GameObject If/Else*.

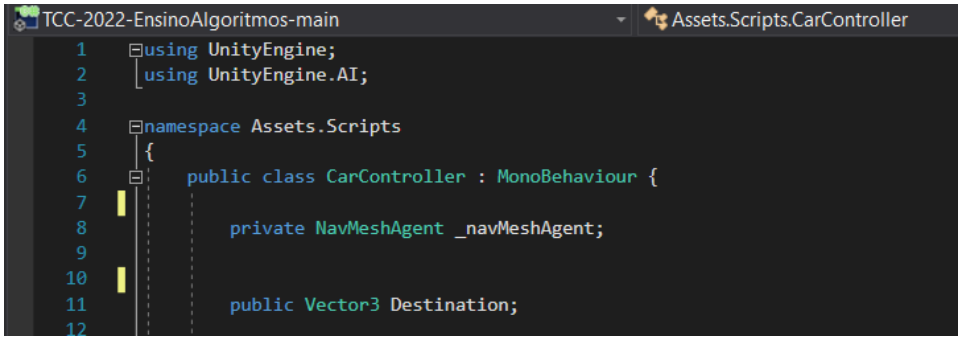

Figura 22. Script CarController - Parte 1 do código

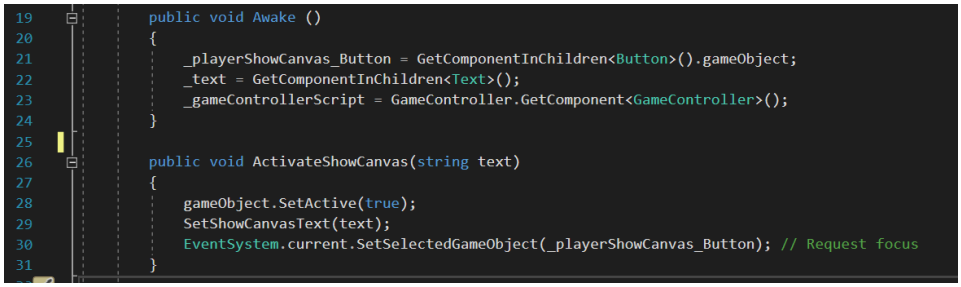

Figura 41. ShowCanvasController - Parte 2 do código

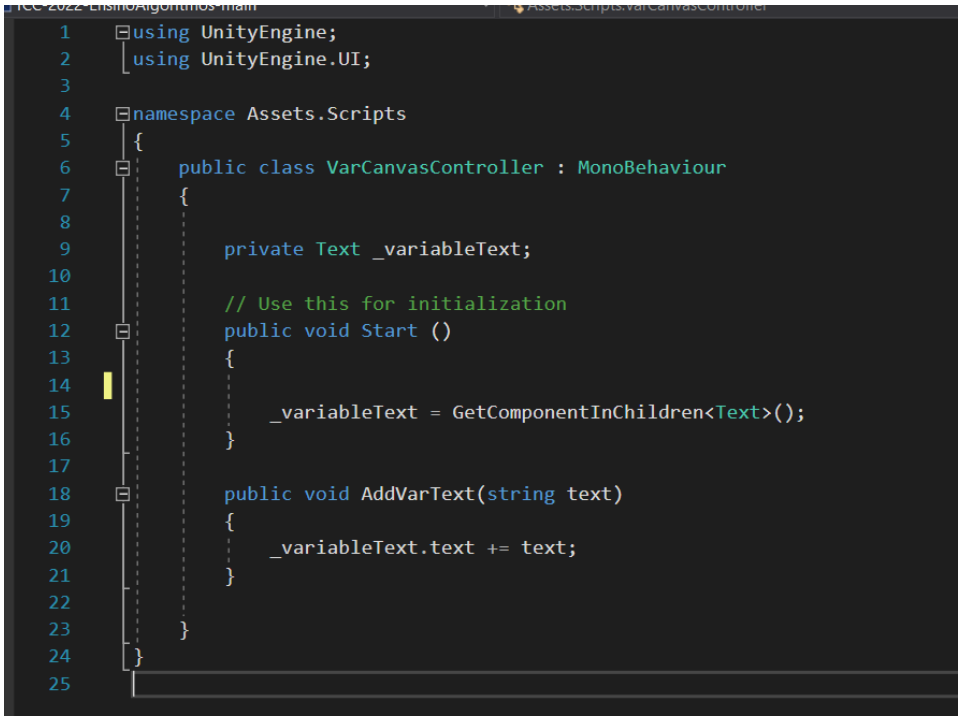

Figura 49. VarCanvasController

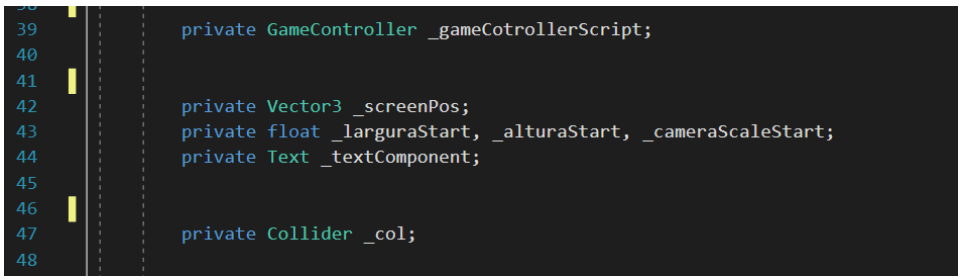

Figura 51. IfElseScript - Parte 2 do código

A representação do método *Start* ocorre na figura 52, o qual nas linhas 57 e 58 definem no componente *Text* o seu conteúdo, na linha 61 pega a referência do *script GameController* e na linha 64 pega a referencia do *collider*.

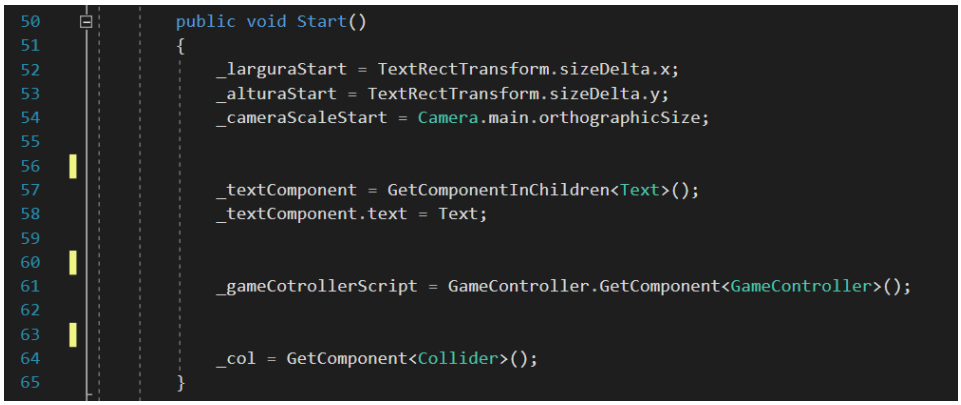

Figura 52. IfElseScript - Parte 3 do código

O método representando na imagem 53, ajusta a posição dos *canvas* de texto na tela na linha 70 e na linha 74 ajusta a escala dos campos *canvas* de texto.

Figura 53. IfElseScript - Parte 4 do código

Na linha 78 da figura 54 temos o método que irá comunicar-se com o interpretador, o qual une o código da inicialização com um retorno da expressão sendo avaliada e envia para o *Carpilator* interpretar, obtendo o resultado da expressão (verdadeiro ou falso). Já na linha 85, o método é chamado toda vez que passa em um *IF/Else*. Esse método é responsável por gerar um código-fonte de inicialização das variáveis a partir do nome e seus valores.

```
78          public bool Evaluate()
79          {
80              var code = string.Join(Environment.NewLine, InitializeVariables(), $"return {Text};");
81              return new Carpilator(code, new CCsharp()).Run<bool>();
82          }
83
84
85          private string InitializeVariables()
86          {
87              var symbolTable = ListaDeVariaveis.Select(x => new { Variavel = x, Valor =
88                  _gameCotrollerScript.GetValorDeVariavel(x) });
89              var varInitializations = symbolTable.Select(x => $"int {x.Variavel} = {x.Valor};");
90              return string.Join(Environment.NewLine, varInitializations);
91          }
92
```

Figura 54. IfElseScript - Parte 5 do código

Por fim, na figura 55 temos o método que é chamado, toda vez que um *collider* invade o *collider* desse objeto, se o resultado for verdeiro lê-se o código *If*, caso ao contrário vai para o else e a linha 108 é responsável por desativar o *collider*.

```
92
93          public void OnTriggerEnter(Collider other)
94          {
95
96              if ( Evaluate() )
97              {
98                  _gameCotrollerScript.SetHightlightTextColor(255f, 255f, 0f);
99                  _gameCotrollerScript.MoveHightlightToPosition(HightlightPositionX, HightlightPositionY,
100                     HightlightPositionSize);
101                 BlockPositionFalso.GetComponent<NavMeshObstacle>().enabled = true;
102             }
103             else
104             {
105                 _gameCotrollerScript.SetHightlightTextColor(0f, 0f, 255f); // Cor azul
106                 _gameCotrollerScript.MoveHightlightToPosition(ElseHightlightPositionX, ElseHightlightPositionY,
107                     ElseHightlightPositionSize);
108                 BlockPositionVerdadeiro.GetComponent<NavMeshObstacle>().enabled = true;
109             }
110             _col.enabled = false;
111         }
112     }
```

Figura 55. IfElseScript - Parte 6 do código

5. Conclusão

O objetivo deste artigo foi alcançado com êxito, entretanto a etapa três da metodologia que abordava o teste de qualidade do jogo desenvolvido nos critérios de usabilidade e funcionalidade e a confecção de um relatório apontando as experiências dos alunos envolvidos não foi realizado devido a falta de tempo hábil para a conclusão deste trabalho.

Com o intuito de identificar as principais dificuldades da turma do semestre 2022/1 do IFSC Câmpus Lages da disciplina de programação no curso de Ciência da Computação e também averiguar se os discentes considerariam uma outra forma de ensino-aprendizagem diferente da que se tem hoje, foi realizada a aplicação de um questionário.

O objetivo de desenvolver um interpretador independente foi alcançado, tendo sido seu desenvolvimento e sua integração com o jogo *Unity* finalizados durante o trabalho, possibilitando assim mais flexibilidade para a criação de outros níveis com estruturas de programação diferentes, estruturas de repetição ou funções por exemplo, e até mesmo a criação de níveis dinâmicos, onde o usuário poderia digitar o código que seria interpretado, gerando assim uma

estrada para aquele código, o que traria uma dinâmica completamente nova e muito útil para o aprendizado.

Em possíveis trabalhos futuros poderá ser implementando melhorias e ampliações como novas estruturas de código: funções, laços de repetição e entre outros, e também será necessário aplicar testes para que o jogo digital seja validado.

Referências

Andrade, R. et al. (2016). Uma proposta de oficina de desenvolvimento de jogos digitais para ensino de programação. *Anais dos Workshops do Congresso Brasileiro de Informática na Educação*, 5(1):1127.

da Silva, F. R. et al. (2016). Desenvolvimento de jogos na plataforma unity. *RE3C-Revista Eletrônica Científica de Ciência da Computação*, 11(1).

De Andrade, M. S. (2019). *Adobe Photoshop CC*. Editora Senac São Paulo.

Grossmann, L. O. (2019). Ti precisa de 420 mil novos profissionais até 2024. Disponível em: `https://brasscom.org.br/ti-precisa-de-420-mil-novos-profissionais-ate-2024/`. Acesso em: 14 abr 2022.

Monclar, R. S., Silva, M. A., e Xexéo, G. (2018). Jogos com propósito para o ensino de programação. *Anais do XVII Simpósio Brasileiro de Jogos e Entretenimento Digital–SBGames*, pages 1132–1140.

Overflow, S. (2021). Developer survey 2021. Disponível em: `https://insights.stackoverflow.com/survey/2021`. Acesso em: 26 jun. de 2022.

Pantaleão, E. et al. (2017). Uma abordagem baseada no ambiente robocode para ensino de programação no ensino médio. *Revista Brasileira de Informática na Educação*, 25(03):95.

Portugal, R. L. Q. e do Prado Leite, J. C. S. (2015). A transparência do github para uso de artefatos como fontes de informação na engenharia de requisitos. *WTRANS*, 15:1–6.

Silva, R. et al. (2018a). Panorama da utilização de jogos digitais no ensino de programação no nível superior na Última década: Uma revisão sistemática da literatura. *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação - SBIE)*, 29(1):535.

Silva, R. et al. (2018b). Panorama da utilização de jogos digitais no ensino de programação no nível superior na Última década: Uma revisão sistemática da literatura. *Brazilian Symposium on Computers in Education (Simpósio Brasileiro de Informática na Educação - SBIE)*, 29(1):535.

Trello (2022). O trello ajuda os times a agilizarem o trabalho. Disponível em: `https://trello.com/home/`. Acesso em: 20 jun. de 2022.

Wagner, B. et al. (2022). Um tour pela linguagem c#. Disponível em: `https://docs.microsoft.com/pt-br/dotnet/csharp/tour-of-csharp/`. Acesso em: 20 jun. de 2022.

Zanetti, H. e Oliveira, C. (2015). Práticas de ensino de programação de computadores com robótica pedagógica e aplicação de pensamento computacional. *Anais dos Workshops do Congresso Brasileiro de Informática na Educação*, 4(1):1236.