

GUILHERME RAMALHO COSTA

**DESENVOLVIMENTO DE UM JOGO SÉRIO PARA A
REABILITAÇÃO DA MARCHA HUMANA**



UNIVERSIDADE FEDERAL DE UBERLÂNDIA
FACULDADE DE ENGENHARIA MECÂNICA

2018

GUILHERME RAMALHO COSTA

**DESENVOLVIMENTO DE UM JOGO SÉRIO PARA A REABILITAÇÃO
DA MARCHA HUMANA**

Monografia apresentada ao Curso de Graduação em Engenharia Mecatrônica da Universidade Federal de Uberlândia, como parte dos requisitos para a obtenção do título de ENGENHEIRO MECATRÔNICO.

Área de Concentração: Robótica.

Orientador: Prof. Dr. Rogério Sales Gonçalves

UBERLÂNDIA

2018

*A todos aqueles que, um dia, se
beneficiarão deste projeto.*

AGRADECIMENTOS

Agradeço a Deus por me proporcionar forças para continuar neste caminho até o final.

Meus pais, Claudio Marcio da Costa e Pauleni de Fátima Ramalho e Costa pelo apoio incondicional durante todo o processo. Agradeço as minhas irmãs, Amanda Ramalho Costa e Ana Cláudia Ramalho Costa e minha namorada Dayane Karine Inácio pelo suporte e confiança. E a minha família, que acreditaram que eu iria concluir com sucesso.

A Universidade Federal de Uberlândia e à Faculdade de Engenharia Mecânica pelo suporte e garantia de ensino de qualidade durante o curso.

Agradeço ao Professor Dr. Rogério Sales Gonçalves pela orientação e direcionamento para realização do trabalho em questão.

Agradeço ao Laboratório de Automação e Robótica, seus integrantes e sua infraestrutura, que me proporcionaram as condições necessárias para a realização deste projeto.

As pessoas especiais durante esta caminhada, meus amigos próximos, que caminharam comigo vencendo todos os obstáculos.

Ao curso de Engenharia Mecatrônica, alunos, professores e profissionais, meus sinceros agradecimentos.

RAMALHO COSTA, G. **Desenvolvimento de um jogo sério para a reabilitação da marcha humana**. 2018. 130f. Monografia de Conclusão de Curso (Bacharel) Universidade Federal de Uberlândia, Uberlândia, 2018

RESUMO

O objetivo deste trabalho é o desenvolvimento de um jogo eletrônico sério para reabilitação com o intuito de complementar a fisioterapia tradicional em pacientes pós-AVE ajudando-os a recuperarem-se da deficiência da marcha. O desenvolvimento foi feito utilizando o Microsoft Kinect, o motor de jogos Unity 3D e uma esteira ergométrica para criar um jogo sério com os parâmetros necessários para a reabilitação de marcha para estes pacientes. O aumento da motivação em pacientes faz com que estes permaneçam por mais tempo e que realizem melhor os protocolos de reabilitação, uma vez que o jogo auxilia na resposta visual e *feedback* do tratamento. Assim, tem-se que a utilização dos jogos sérios pode ser uma ferramenta que trará um resultado positivo no processo de reabilitação da marcha em pacientes pós-AVE.

Palavras Chave: Reabilitação da marcha. Jogos sérios. Fisioterapia. Reabilitação pós-AVE

RAMALHO COSTA, G. **Development of a serious game for human gait rehabilitation**. 2018. 130p. Course Conclusion Monograph (Bachelor) Federal University of Uberlândia, Uberlândia, 2018

ABSTRACT

The purpose of this study is the development of a serious electronic game for human gait rehabilitation to complement traditional physiotherapy in post-stroke patients to help them to recover from the deficient gait. The development was made using the Microsoft Kinect, the game engine Unity 3D and a treadmill to build a serious game with needed parameters to a gait rehabilitation for these patients. Increased motivation in patients causes them to stay longer and to better perform rehabilitation protocols, since serious game can help in the visual response and feedback of treatment. Thus, it has been observed that the use of serious games can be a tool that will bring a positive result in the process of rehabilitation of gait in post-stroke patients.

Key Words: Gait rehabilitation. Serious games. Physiotherapy. Stroke rehabilitation.

LISTA DE FIGURAS

Figura 2.1 - Valores aproximados das subfases e fases da marcha humana de pessoas normais (LEITE; MEIJA, 2013) (modificado).....	15
Figura 2.2 - O desenvolvimento dos jogos sérios através dos jogos (ZYDA, 2005) (modificado).....	17
Figura 2.3 - A segunda versão do sensor Kinect.....	18
Figura 2.4 - Adaptador do sensor Kinect para conexão USB	19
Figura 2.5 - Interface do Unity 3D evidenciando as principais views: <i>Scene</i> , <i>Game</i> , <i>Hierarchy</i> , <i>Project</i> e <i>Inspector</i>	20
Figura 2.6 - Janela <i>Inspector</i> de um objeto de jogo	22
Figura 2.7 - Exemplo de componentes para serem adicionados na opção "Add Component"	23
Figura 2.8 - Exemplo de código inicial dos <i>scripts</i> criados no software <i>MonoDevelop</i>	24
Figura 3.1 - Diagrama de utilização do jogo desenvolvido	26
Figura 3.2 - Principal interface de usuário da cena "MenuPrincipal"	28
Figura 3.3 - Representação do objeto de jogo do tipo <i>Canvas</i> dentro da janela <i>Hierarchy</i>	28
Figura 3.4 - Menu de ajuda ao usuário.....	29
Figura 3.5 - Menu de informações sobre o jogo ao usuário	29
Figura 3.6 - Imagem da tela de carregamento entre cenas.....	30
Figura 3.7 - A janela <i>Hierarchy</i> mostra os objetos de jogo da cena atual	30
Figura 3.8 - Objeto de jogo que representa o terreno desenvolvido para o cenário..	31
Figura 3.9 - Tela principal da cena de inserção de usuário	32
Figura 3.10 - Exemplo de relatório gerado após cada partida.....	33
Figura 3.11 - Tela principal da cena de jogo	34
Figura 3.12 - Detalhe do modelo utilizado para o avatar	35
Figura 3.13 - Hierarquia da construção do avatar	36
Figura 3.14 - Pontos identificados pelo sensor Kinect para a criação do esqueleto e das referências para o avatar.....	37
Figura 3.15 - Visão da ponte dentro do jogo	38
Figura 3.16 - Visão em detalhe do obstáculo dentro do jogo	40

Figura 3.17 - Efeito de explosão de estrelas que acontece quando o usuário toca nos obstáculos	40
Figura 3.18 - Painel <i>Canvas</i> de informação de desempenho do jogador	42
Figura 3.19 - Configuração "On Click ()" dentro do componente "Button (Script)"	43
Figura 3.20 - Caixa de seleção das cenas que fazem parte do jogo em construção	44
Figura 3.21 - Imagem gerada pelo sensor Kinect para identificação do corpo humano e alguns pontos de articulação	45
Figura 3.22 - Representação do gatilho (em verde) dentro do mundo do jogo (em preto)	47
Figura 4.1 - Gráfico representativo das respostas do teste de experiência de jogo ..	52

LISTA DE SÍMBOLOS E ABREVIACÕES

UFU – Universidade Federal de Uberlândia

AVE – Acidente Vascular Encefálico

OMS – Organização Mundial da Saúde

AVC – Acidente Vascular Cerebral

IDE – *Integrated Development Environment*

UI – *User Interface*

GEQ – *Game Experience Questionnaire*

SUMÁRIO

CAPÍTULO I - INTRODUÇÃO.....	11
1.1. Objetivos.....	12
1.1.1. Objetivo geral	12
1.1.2. Objetivos específicos.....	12
1.2. Justificativa	13
CAPÍTULO II - FUNDAMENTAÇÃO TEÓRICA.....	14
2.1. Marcha humana.....	14
2.2. Acidente Vascular Encefálico	15
2.2.1. Alterações na marcha humana após AVE.....	16
2.3. Protocolos de reabilitação	16
2.4. Jogos sérios.....	17
2.5. Microsoft Kinect.....	18
2.5.1. Adaptador para USB	18
2.6. Unity 3D	19
2.6.1. Interface	19
2.6.2. Importação de assets e packages	21
2.6.3. Desenvolvimento de jogos	21
CAPÍTULO III - METODOLOGIA.....	26
3.1. Modelagem do jogo	26
3.2. O jogo.....	27
3.2.1. Interface menu principal	27
3.2.2. Interface de inserção de paciente.....	32
3.2.3. Interface principal do jogo	34
3.2.4. Avatar	34
3.2.5. Ponte	38
3.2.6. Obstáculos	39
3.2.7. Sistema de pontuação e tempo de partida	41
3.3. Métodos de controle	43
3.3.1. Menus de utilização.....	43

3.3.2.	Marcha do avatar.....	44
3.3.3.	Controle de alterações	46
3.4.	Testes de experiência – <i>Game Experience Questionnaire</i> (GEQ).....	48
3.4.1.	Usuários	49
3.4.2.	Procedimento para os testes	49
CAPÍTULO IV - RESULTADOS E DISCUSSÕES		51
CAPÍTULO V - CONCLUSÕES		54
REFERÊNCIAS BIBLIOGRÁFICAS		56
APÊNDICES		59

CAPÍTULO I

INTRODUÇÃO

O Acidente Vascular Encefálico (AVE) é, segundo dados de 2016, mundialmente, a principal causa de morte e incapacidade em adultos. Estima-se que cerca de 57% dos sobreviventes do AVE necessitarão de cuidados contínuos (KARENY et al., 2016).

Segundo o Relatório Mundial sobre a Deficiência, publicado pela Organização Mundial da Saúde (OMS) juntamente com o Banco Mundial em 2011, mais de 1 bilhão de pessoas ao redor de todo o mundo convivem com algum tipo de deficiência, tornando este assunto uma questão de direitos humanos destacando, entre outros, a reabilitação (BRASIL, 2013).

O AVE refere-se ao desenvolvimento rápido de sinais clínicos de distúrbios que podem ser locais ou globais da função encefálica, com duração maior ou igual a 24 horas com origem vascular que irá provocar alterações nos planos cognitivos e sensório-motor, variando a gravidade de acordo com a área e a extensão da lesão no encéfalo (BRASIL, 2013).

Após ser acometido por um AVE, os pacientes sofrem perdas neurológicas que irão interferir na cognição, comunicação, no emocional e, principalmente, no sistema neuromuscular. As principais perdas nestes indivíduos se dão no padrão anormal da marcha, hemiparesia, perda do equilíbrio e espasticidade muscular (BELDA-LOIS et al., 2011; LOTERIO, 2015).

Embora a maioria dos pacientes pós-AVE consigam caminhar novamente sem auxílio, eles poderão não ter a destreza suficiente para executar suas atividades diárias novamente. A prática das habilidades motoras de locomoção na intenção de melhorar o desempenho da marcha é um dos principais pontos na reabilitação pós-AVE (FLANSBJER et al., 2005).

Para a maioria das lesões, a reabilitação é um processo longo e árduo, onde os pacientes e os profissionais enfrentam diversos problemas com origens variadas, como a motivacional. Assim, o uso de tecnologias que tornam estes processos menos metódicos e

cansativos faz com que os pacientes se sintam imersos no tratamento, gerando ainda mais resultados (WEISS et al., 2004).

A utilização do Microsoft Kinect® para a reabilitação já é uma realidade (MOUSAVI HONDORI; KHADEMI, 2014). Entretanto, a utilização de jogos específicos para a reabilitação conhecidos como *serious games* (jogos sérios, na tradução literal) ainda é pouco difundida (GARCIA MARIN et al., 2011).

Os *serious games* já são estudados e provam a eficiência nos tratamentos de reabilitação em quesitos motivacionais. O que falta é uma ampla utilização por estas tecnologias de forma eficiente nas terapias pós-AVE (MA; BECHKOUM, 2008).

A utilização do Kinect® juntamente com a plataforma Unity 3D, um programa de desenvolvimento para jogos com suporte para Kinect, se deu pela facilidade de utilização dos dois componentes juntos, uma vez que os mesmos já foram amplamente utilizados para jogos de entretenimento além de possuírem o custo de desenvolvimento baixo quando comparado a outros meios (LANGE et al., 2011).

1.1. Objetivos

1.1.1. Objetivo geral

O objetivo geral deste trabalho é o desenvolvimento de um jogo sério (*serious game*), utilizando o Microsoft Kinect® e o motor de jogos Unity 3D, com o intuito de ser utilizado juntamente com a fisioterapia tradicional para a reabilitação da marcha humana em pacientes pós-AVE. Desta forma, deverá ser um recurso adicional e não substituto no processo de terapia destes pacientes.

1.1.2. Objetivos específicos

Os objetivos específicos para o trabalho em questão são:

- Planejamento do jogo sério;
- Estudo do desenvolvimento de jogos;
- Desenvolvimento do cenário;
- Adaptação do avatar;
- Desenvolvimento da ligação entre a velocidade do avatar e a velocidade da esteira;
- Desenvolvimento do jogo e do menu iniciar;
- Desenvolvimento da tela de inserção dos dados do usuário;

- Determinação da forma de pontuação dentro do jogo;
- Realizar um questionário de opinião sobre o jogo com pessoas saudáveis;

1.2. Justificativa

Ao usar tecnologias dos diversos tipos para incrementar a terapia da reabilitação da marcha humana, tem-se uma melhora significativa da aplicação dos pacientes no decorrer das seções. A utilização delas baseia-se nas premissas da neuroplasticidade, buscando além de atuar na movimentação repetitiva, explorar pontos da cognição do paciente (FLORES et al., 2008).

Com o passar do tempo, os pacientes vão abandonando a terapia devido a perda da motivação para acompanhamento da mesma (MOYÀ ALCOVER et al., 2011). Assim, este trabalho se justifica no interesse de atingir melhor a população que necessita da reabilitação de forma a reduzir o número de pessoas que abandonam a terapia e, também, aumentar a força e o desejo de realizar da melhor forma os movimentos requeridos pelo profissional.

CAPÍTULO II

FUNDAMENTAÇÃO TEÓRICA

Neste capítulo, vamos apresentar as informações básicas para melhor entendimento do trabalho, tais como definições da marcha humana, AVE, protocolos de reabilitação, jogos sérios, Microsoft Kinect® e o motor de jogos Unity 3D.

2.1. Marcha humana

A marcha humana é composta por uma performance repetida e alternada dos membros inferiores, mantendo uma postura estável de sustentação, além de mobilização articular, ação muscular seletiva, forças de reação com o solo, movimentos associados dos membros inferiores, superiores, tronco, cabeça e olhos, oscilações suaves do centro de gravidade do ser humano, e eficiência em relação ao gasto energético (OTTOBONI et al., 2002; IWABE et al., 2008).

Em indivíduos normais, a marcha é dividida em duas fases, sendo elas a fase de apoio, que corresponde a 60% do total da passada e a fase de balanço, que corresponde a 40%. A fase de apoio fornece a sustentação, estabilidade e também a propulsão. Já o balanço fornece o avanço e o afastamento dos membros. As subfases e seus valores aproximados são mostradas na Figura 2.1.

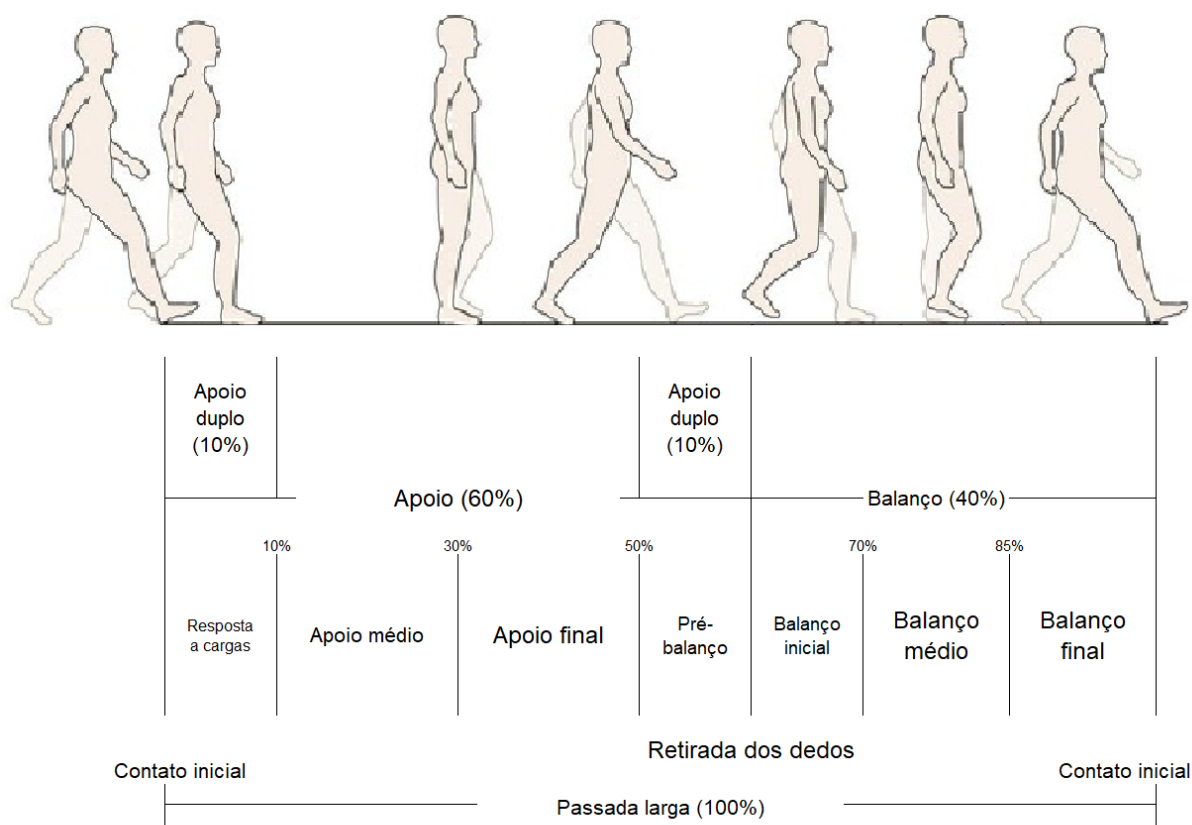


Figura 2.1 - Valores aproximados das subfases e fases da marcha humana de pessoas normais (LEITE; MEIJA, 2013) (modificado).

2.2. Acidente Vascular Encefálico

O AVE refere-se ao desenvolvimento rápido de sinais clínicos de distúrbios que podem ser locais ou globais da função encefálica, com duração maior ou igual a 24 horas com origem vascular que irá provocar alterações nos planos cognitivos e sensório-motor, variando a gravidade de acordo com a área e a extensão da lesão no encéfalo (BRASIL, 2013).

O termo evoluiu ao longo do tempo, de forma a acrescentar lesões que são causadas por outros distúrbios, como os hemodinâmicos e os da coagulação, deixando de ser utilizado o termo Acidente Vascular Cerebral (AVC) em diversos casos (SEGURA et al., 2008).

Após ser acometido por um AVE, os pacientes sofrem perdas neurológicas que irão interferir em diversas partes do corpo. As principais perdas nestes indivíduos se dão no padrão anormal da marcha, hemiparesia, perda do equilíbrio e espasticidade muscular (BELDA-LOIS et al., 2011; LOTERIO, 2015).

2.2.1. Alterações na marcha humana após AVE

A marcha hemiparética, ou seja, a marcha humana em pacientes que possuem perda de sensibilidade de um dos lados do corpo, não é coordenada nem rítmica, consumindo alto teor de energia do ser humano, além de não possuir equilíbrio necessário (OTTOBONI et al., 2002).

Embora a maioria dos pacientes pós-AVE consigam caminhar novamente sem auxílio, eles poderão não ter a destreza suficiente para executar suas atividades diárias novamente. A prática das habilidades motoras de locomoção na intenção de melhorar o desempenho da marcha é um dos principais pontos na reabilitação pós-AVE (FLANSBJER et al., 2005).

As principais deficiências na marcha entre pacientes pós-AVE consistem em alterações na velocidade, balanço, comprimento do passo, cadência, simetria, desajustes quanto à postura e equilíbrio, entre outras, uma vez que a deficiência da marcha varia de acordo com o grau e o local da lesão sofrida pelo paciente (OTTOBONI et al., 2002).

2.3. Protocolos de reabilitação

É de senso comum que os protocolos de reabilitação possuem como elemento principal a prática bem orientada, definida e repetitiva dos movimentos (SIGN, 2010).

Na literatura, existem muitos protocolos de tratamento para pacientes com alterações na marcha provenientes de AVEs. A maioria deles possui como objetivo levar a melhora para o paciente de forma que o mesmo consiga uma independência em atividades rotineiras.

Para (SILVA, 2010), o processo de reabilitação possui 6 parâmetros principais:

1. Reconhecimento, gestão e prevenção de complicações e de outras doenças que podem aparecer ao mesmo tempo;
2. Terapia para independência no máximo de atividades;
3. Facilitação da capacidade do indivíduo e da família do mesmo a lidar com a situação atual e se adaptarem para melhor convivência;
4. Prevenção de deficiências secundárias por meio do incentivo a reintegração social (família, comunidade, etc.);
5. Reforço da qualidade de vida em relação às deficiências remanescentes; e
6. Prevenção de um segundo episódio.

Estudos clínicos mostram que os processos de reabilitação devem ser iniciados assim que a condição do paciente permita, uma vez que tal atitude leva a melhoria dos aspectos físicos e psicológicos do paciente (SHAH et al., 1990).

2.4. Jogos sérios

Os dicionários buscam definir videogame como jogo que são mostrados em telas de computadores ou televisões que permitem que o usuário manipule eletronicamente as ações destas imagens, resultando em acontecimentos específicos (“Videogame | Michaelis On-line”).

Entretanto, pode-se definir videogame como um torneio mental, jogado com um computador seguindo regras de diversão, recreação ou participação. E, assim, define-se *serious games* como um torneio mental, jogado com um computador seguindo regras específicas, que usa entretenimento para fornecer treinamento, educação, saúde, conhecimento político, e objetivos estratégicos de comunicação (ZYDA, 2005).

Dessa forma, os jogos sérios (*serious games*) compreendem mais do que apenas história, artes e computação. Eles envolvem a chamada pedagogia: atividades que possuem como objetivo a ideia de instruir e de educar, o que irá gerar conhecimento ou habilidades específicas. No entanto, o entretenimento vem em primeiro lugar e, em seguida, a pedagogia acontece quando o entretenimento funciona (ZYDA, 2005).

A Figura 2.2 mostra a relação entre os jogos comuns com os jogos sérios e como eles estão interligados.

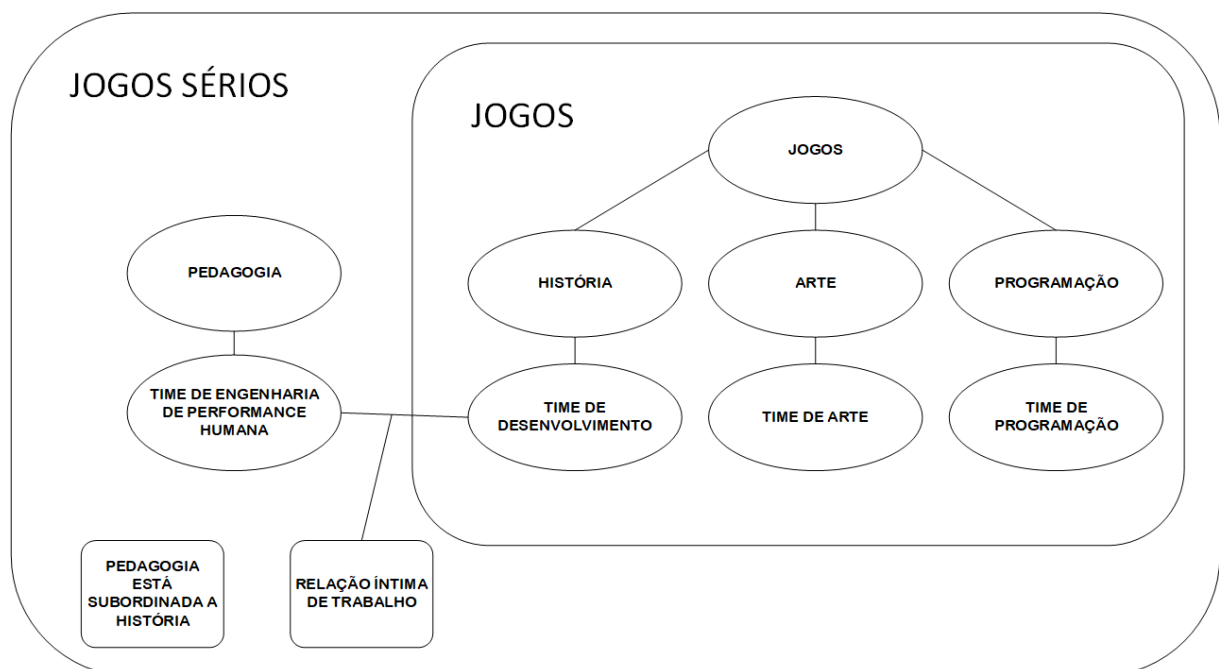


Figura 2.2 - O desenvolvimento dos jogos sérios através dos jogos (ZYDA, 2005) (modificado).

2.5. Microsoft Kinect

O Microsoft Kinect, ou apenas Kinect, é um dispositivo com diversos sensores instalados num mesmo hardware. A versão utilizada neste projeto é a segunda e pode ser vista na Figura 2.3.



Figura 2.3 - A segunda versão do sensor Kinect.

Fonte: <https://goo.gl/WZ4wtR> (Acesso em: 01/07/2017).

Seus itens mais notáveis são: sensor de profundidade, câmera colorida, um sensor infravermelho e um microfone. Através da junção destes componentes, o Kinect consegue processar uma captura em 3D de todo o corpo, reconhecimento de face e reconhecimento de voz (ZHANG, 2012).

A escolha do Kinect para compor este trabalho se dá pela facilidade de programação que o mesmo possui, uma vez que seu desenvolvedor, a Microsoft, distribui um kit de desenvolvimento de software (*Software Development Kit* – SDK) gratuitamente em sua página na internet (MICROSOFT).

2.5.1. Adaptador para USB

Para ser utilizado com um computador, é necessário a utilização de um adaptador específico para o sensor Kinect, mostrado na Figura 2.4.

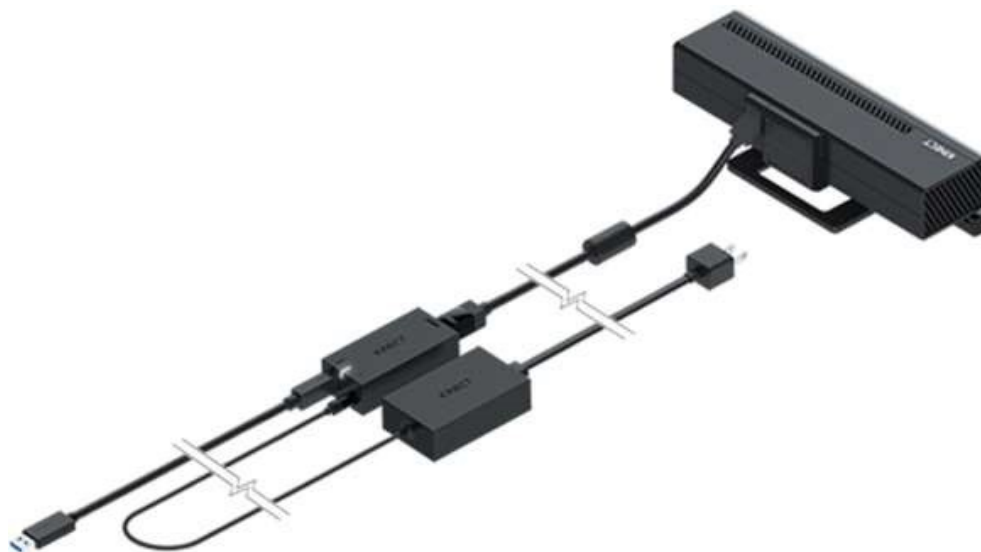


Figura 2.4 - Adaptador do sensor Kinect para conexão USB.

Fonte: <https://goo.gl/9rhik2> (Acesso em: 01/07/2017).

Para a utilização deste dispositivo, é necessária a utilização de uma porta USB *SuperSpeed*, conhecida também como USB 3.0.

2.6. Unity 3D

O Unity 3D é um motor de jogos (*game engine*) amplamente utilizada na criação de jogos 2D e 3D. Estas plataformas devem possuir diversos módulos e funcionalidades que as fazem ser adaptáveis em diferentes plataformas sem muitas complicações, uma vez que um jogo possui campos diversos onde diferentes áreas do conhecimento devem ser aplicadas (PASSOS et al., 2009).

Além de ser uma ferramenta bastante completa e possuir diversos níveis de aplicações, o Unity 3D é uma plataforma que possui uma versão gratuita para desenvolvimento de jogos com determinadas limitações que não influenciam no desenvolvimento desta pesquisa (LANGE et al., 2011).

2.6.1. Interface

O Unity 3D possui uma interface incrivelmente simples e amigável que torna o desenvolvimento do jogo seu objetivo principal sem muitas dificuldades. Possui em sua interface diversas janelas voltadas cada uma delas para uma atividade diferente, organizando-as de maneira efetiva e simplificada.

Estas janelas são chamadas de *views* e são exemplificadas na Figura 2.5.

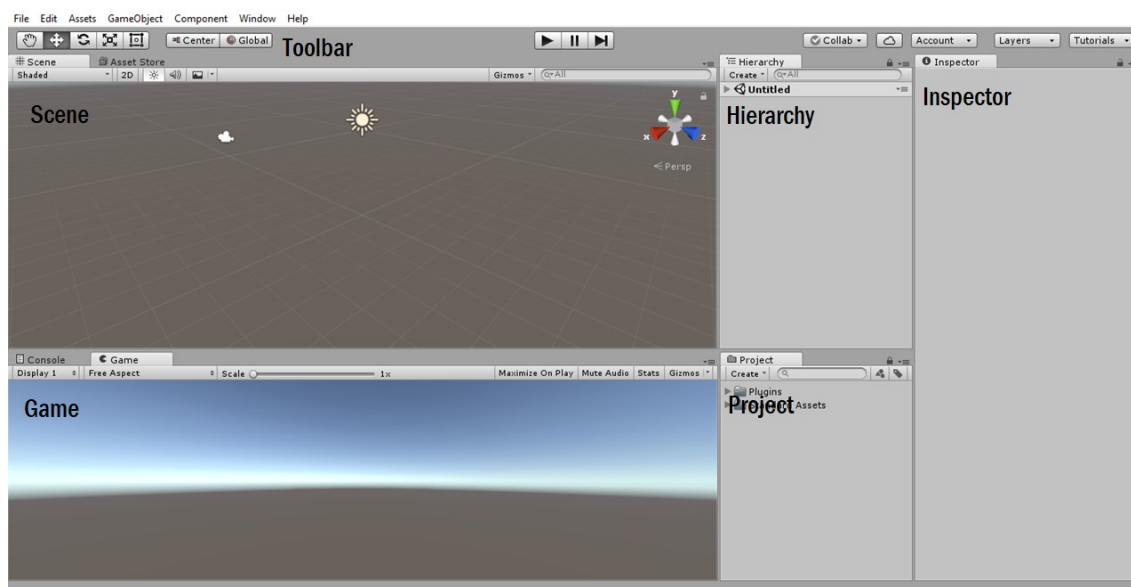


Figura 2.5 - Interface do Unity 3D evidenciando as principais views: *Scene*, *Game*, *Hierarchy*, *Project* e *Inspector*.

A região denominada por *Toolbar* não é necessariamente uma das *views* do Unity 3D. Esta possui as funções e características de quaisquer outras barras de ferramentas, contendo caminhos e atalhos para determinadas tarefas, além de possuir o mecanismo de ativação da simulação do jogo em desenvolvimento.

A janela *Scene* é a principal maneira de manipular os elementos visuais do jogo em desenvolvimento, uma vez que é nesta janela que estes são posicionados e orientados de acordo com a necessidade do jogo.

A janela *Hierarchy* exibe os elementos que estão presentes na janela *Scene* que está sendo editada naquele momento. É através desta janela que podemos organizar e visualizar a hierarquia dos elementos utilizados até então na criação de determinada cena.

Já a janela *Inspector* é onde, ao selecionar um dos elementos da cena, tem-se acesso as configurações do mesmo, assim como seus atributos. É nesta janela que os componentes da cena são configurados para que os mesmos possam trabalhar da forma necessária.

Na janela *Game* tem-se uma representação final do jogo em desenvolvimento. Nesta janela pode-se ter a prévia do comportamento dos elementos que constituem a cena em edição, atendendo rapidamente às mudanças de parâmetros que possam ser feitas na janela de inspeção (*Inspector*).

Por fim, a janela *Project* é similar a janela *Hierarchy*, diferindo desta nos arquivos que ela possui e organiza. É na janela *Project* que se tem acesso aos diversos arquivos que compõem o jogo completo, podendo ser facilmente alterado de um arquivo para outro. A principal função desta janela é garantir a integridade dos arquivos do jogo em desenvolvimento, uma vez que, ao modificar-se os arquivos dentro da plataforma Unity 3D,

estes serão sempre analisados pelo todo através do motor de jogos (PASSOS et al., 2009; MACHADO, 2015).

2.6.2. *Importação de assets e packages*

A importação de *assets* e *packages* consiste em utilizar outros arquivos que podem ser desenvolvidos em outras plataformas, como elementos 3D desenvolvidos em softwares específicos para tal. Estes arquivos são utilizados nas mais diversas aplicações dentro do jogo, uma vez que muitos deles podem facilitar o desenvolvimento do jogo ao oferecer métodos e modelos prontos. Um *package* é o conjunto de vários *assets* que podem ou não trabalhar entre si.

O principal *asset* utilizado no jogo em questão foi desenvolvido pelo professor Rumen Filkov, pesquisador da *Vorarlberg University of Applied Sciences*, na Áustria (FILKOV, 2014). Ele consiste em um *package* de demonstrações e *scripts* para manipulação do Kinect. Embora possua diversos outros exemplos, foram utilizados apenas aqueles que tangem a captação dos movimentos do usuário perante o sensor. O mesmo é disponibilizado gratuitamente para a utilização em pesquisas e projetos acadêmicos.

Foram utilizados também outros *assets* que foram obtidos através de *downloads* na *Internet*, como textura para água e para os botões dos menus.

A ponte utilizada no jogo foi feita através do software SolidWorks pelo próprio autor e, então, convertida para poder ser importada pelo motor de jogos Unity 3D para ser utilizada no jogo, a transformando em um *asset*.

Há também a possibilidade de alterações nos *assets* e *packages* importados dentro do próprio motor de jogos. Fazendo com que seja possível o desenvolvimento dos desenhos de base e, então, modifica-los dentro do software para melhor se adequarem as necessidades do jogo.

2.6.3. *Desenvolvimento de jogos*

O desenvolvimento de jogos no Unity 3D, uma vez sendo sua finalidade, é facilitado e feito de forma a abranger diversos tipos de profissionais, desde programadores até artistas visuais. Desta forma, utiliza-se diferentes tipos de desenvolvimento de jogos, sendo possível executar tarefas iguais de formas diferentes, na maioria das vezes, não excludentes entre si.

Nesta seção, será descrito o sistema de *GameObjects* e o sistema de *Scripting*.

2.6.3.1. Sistemas de *GameObjects*

O Unity 3D utiliza uma nova arquitetura de objetos de jogos baseado em composição. Nesta arquitetura, um objeto de jogo é definido através da composição de várias funcionalidades ao mesmo. Estas funcionalidades são componentes do objeto de jogo e, cada funcionalidade, é um componente diferente (PASSOS et al., 2008, 2009).

Estes componentes vão desde *scripts*, texturas, sons até geometrias físicas ou de *trigger*. Desta forma, podemos concluir que os *GameObjects* podem ser qualquer objeto dentro do jogo, podendo ser parte do jogo manipulável, do cenário e, até mesmo, a câmera e a iluminação.

Todos os *GameObjects* possuem, pelo menos, o componente *Transform* que é sua posição, orientação e sua escala dentro do jogo. Como este componente é comum para todos os objetos, ele é utilizado para definição de hierarquias, podendo acessar objetos pais e/ou filhos através do componente *Transform*.

A Figura 2.6 mostra a janela *Inspector* do objeto de jogo “Terreno”. Note que, o objeto “Terreno” só se comporta como um *GameObject Terrain* uma vez que o mesmo possui o componente *Terrain* para dar-lhe esta funcionalidade. Note também que o mesmo possui o componente *Transform* como dito anteriormente e, além destes dois, possui também um componente chamado *Terrain Collider* que dá ao mesmo a funcionalidade de colidir com outros objetos de jogo.

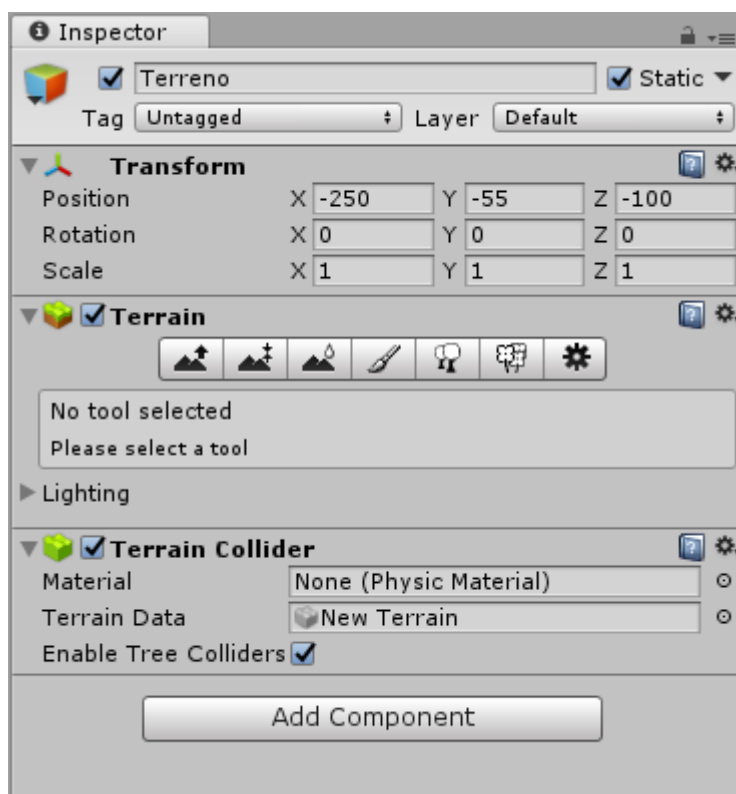


Figura 2.6 - Janela *Inspector* de um objeto de jogo.

Também é possível notar na Figura 2.6 que há um botão *Add Component*. É este botão responsável por adicionar os componentes dentro dos objetos de jogo. A Figura 2.7 mostra parcialmente quais são alguns tipos de componentes que possam ser adicionados a um objeto de jogo.

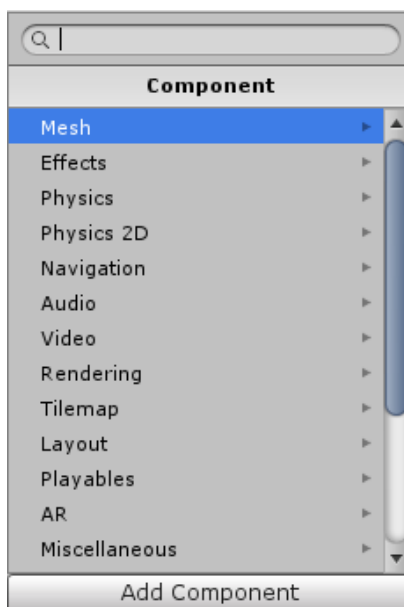


Figura 2.7 - Exemplo de componentes para serem adicionados na opção "Add Component".

Por meio destes conceitos de objeto de jogo e de componentes é possível criar objetos pré-definidos por meio do conceito de *Prefabs* (modelos pré-fabricados). Estes são como modelos que já possuem todos os componentes necessários para caracterizar um objeto de jogo específico.

Os *Prefabs* são muito utilizados quando há a repetição de objetos no jogo, uma vez que, através dos mesmos é possível repetir o mesmo padrão em todos os objetos.

2.6.3.2. Scripting

O *Scripting* do Unity 3D consiste em um modelo bastante flexível e que abrange bastante informações de maneira simples e rápida. Os códigos podem ser escritos por meio de *JavaScript* ou *C#* (*C Sharp*) e não há diferenças entre desempenho entre estas duas linguagens dentro do Unity 3D, podendo ser escolhida aquela que mais se adequa a situação e, também podendo ser utilizada as duas dentro de um mesmo projeto (PASSOS et al., 2009).

Neste projeto foi escolhido a utilização da linguagem *C#* uma vez que a maioria da literatura e informações obtidas nas pesquisas estão utilizando esta linguagem.

Como dito anteriormente, os *scripts* agem como componentes para os objetos de jogos e são adicionados como qualquer outro componente.

Por meio dos *scripts* é possível acessar outros componentes do mesmo objeto de jogo em que o mesmo está fazendo parte, de forma a poder utilizar suas informações ou, até mesmo, modifica-los de acordo com o decorrer do jogo.

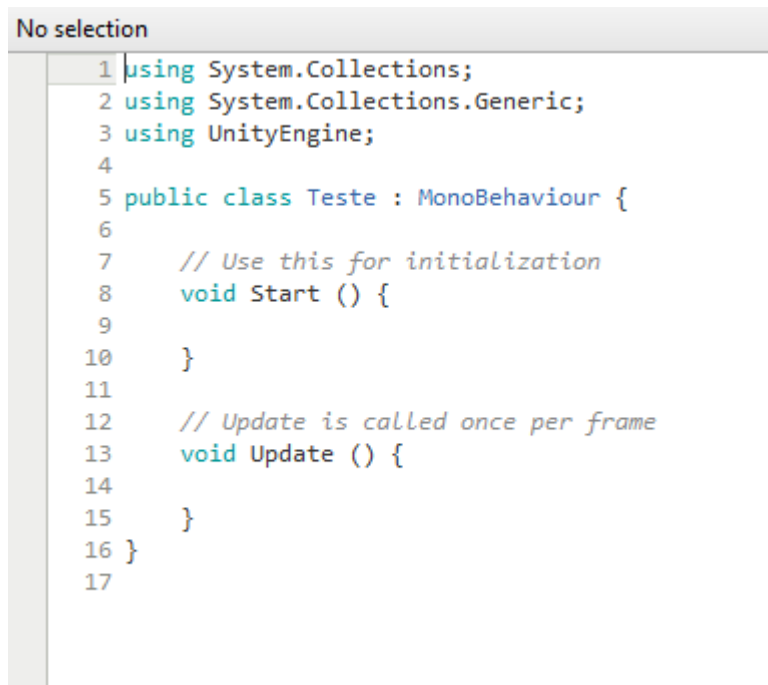
Além disso, como falado anteriormente, por meio do componente *Transform* é possível, por meio das técnicas de *scripting* acessar informações da hierarquia do objeto de jogo.

É possível obter informações também de outros objetos de jogos instanciados na mesma cena através de variáveis públicas, da mesma forma como feito em programação orientada a objetos.

Também é possível obter informações de outras cenas, através de métodos específicos do Unity 3D, o *PlayerPrefs*, desta forma, é possível obter uma comunicação entre diferentes cenas e, assim, passar informações, como pontuações, por exemplo.

O Unity 3D fornece um ambiente de desenvolvimento integrado (IDE), o *MonoDevelop*. Não é obrigatório o desenvolvimento dos códigos neste IDE, uma vez que ele é apenas sugerido pelo Unity 3D, entretanto, a comunicação entre os dois softwares acontece de maneira muito simples, tendo então o desenvolvimento dos jogos facilitado quando se utiliza este IDE no *scripting*.

A Figura 2.8 mostra a tela inicial dos *scripts* feitos no *MonoDevelop*. Estes são bastante intuitivos e, geralmente, possuem dois métodos principais, que são o *Start ()* e o *Update ()*.



```

No selection
1 using System.Collections;
2 using System.Collections.Generic;
3 using UnityEngine;
4
5 public class Teste : MonoBehaviour {
6
7     // Use this for initialization
8     void Start () {
9
10    }
11
12    // Update is called once per frame
13    void Update () {
14
15    }
16 }
17

```

Figura 2.8 - Exemplo de código inicial dos *scripts* criados no software *MonoDevelop*.

O método *Start ()* faz a configuração inicial do *script*, sendo executado uma única vez quando o script inicia sua execução. Já o método *Update ()* será executado novamente a cada frame do jogo.

Existem outros métodos que foram criados pelo autor ou que não são mostrados na Figura 2.8 e foram utilizados. Estes serão explicados dentro de suas respectivas áreas.

CAPÍTULO III

METODOLOGIA

Neste capítulo será apresentado as etapas de desenvolvimento do trabalho, bem como os métodos utilizados para este desenvolvimento e suas justificativas de escolhas.

3.1. Modelagem do jogo

O jogo foi modelado de forma a ser realizado com a supervisão de um profissional com conhecimento na área da terapia e reabilitação que, além de realizar o acompanhamento da terapia, será responsável pela inicialização, entrada de dados e finalização do jogo.

A Figura 3.1 mostra um diagrama de utilização entre o usuário paciente, usuário terapeuta e o jogo.

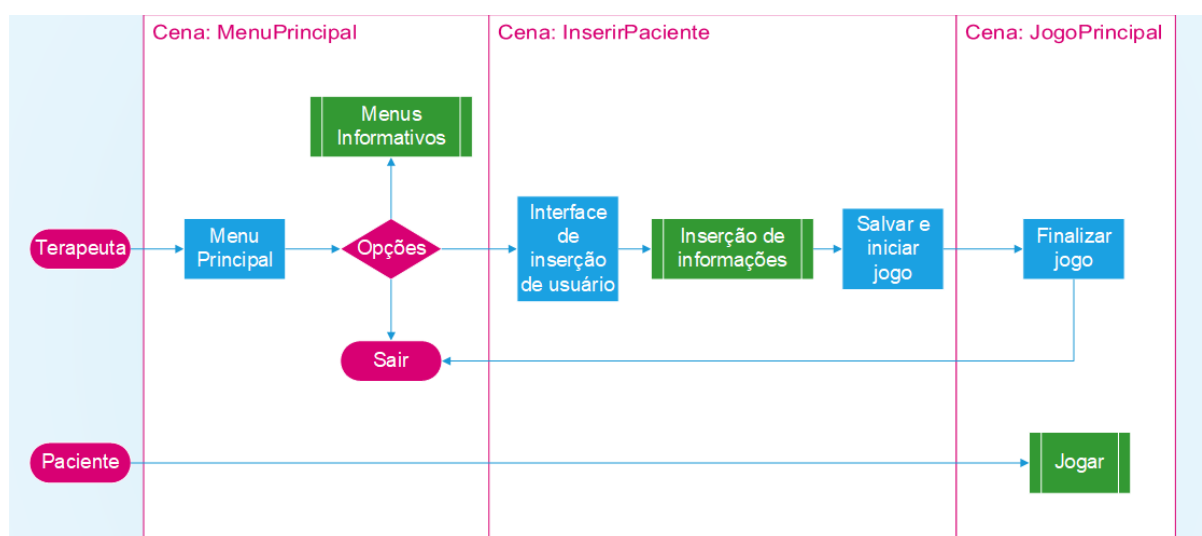


Figura 3.1 - Diagrama de utilização do jogo desenvolvido.

Nesta Figura 3.1 visualizou-se a interação entre os usuários e o jogo, em suas três telas: menu principal, inserção de paciente e ambiente de jogo. As três telas são bem definidas e, possuem uma sequência pré-estabelecida.

No menu principal, o profissional terá acesso a quatro direções: **iniciar**, **ajuda**, **sobre** e **sair**. Nas opções de ajuda e sobre, apenas será mostrado um texto informativo, sem necessidade de interação com o usuário. Já na opção sair, o usuário será levado para fora do jogo.

A opção iniciar levará para a segunda tela, como mostra a Figura 3.1. Nesta segunda tela, a inserção do paciente será descrita em cada um dos campos pelo próprio profissional.

Após o preenchimento das informações nesta tela, a opção será de iniciar o jogo, que irá levar o usuário para a tela de ambiente de jogo.

Nesta tela de ambiente de jogo, é o usuário paciente que irá interagir com o jogo, representado pelo avatar caminhando pela ponte. O mesmo terá os obstáculos para serem tocados e o caminho a ser percorrido na velocidade determinada pela esteira.

Há também, nesta tela, um mostrador dos pontos e do tempo de partida corrido e também um botão de finalização da partida para ser utilizado pelo profissional. O mesmo faz com que o jogo volte para o menu principal e, então, o usuário terapeuta possa sair do jogo conforme as instruções anteriores.

3.2. O jogo

Nesta seção, vamos apresentar todos os aspectos do jogo, de forma a ambientar melhor o leitor neste desenvolvimento. As informações aqui apresentadas serão complementadas com os documentos informados nos apêndices e anexos deste trabalho.

3.2.1. Interface menu principal

A interface do menu principal pode ser vista na Figura 3.2. Ela é composta simplesmente por quatro botões que executam as ações nomeadas em cada um deles.



Figura 3.2 - Principal interface de usuário da cena "MenuPrincipal".

Para a criação de uma tela como a mostrada na Figura 3.2, deve-se utilizar elementos da interface de usuário do Unity 3D, a *UI System*. É através destes componentes que é possível criar telas de interação com o usuário a partir de botões, textos, controles deslizantes, controles de alternância, entre outros.

E, para uma melhor organização destes, é necessário que sejam criados dentro de um objeto de jogo chamado *Canvas*, de forma que os componentes sejam objetos filhos deste painel *Canvas*, como mostrado na Figura 3.3.

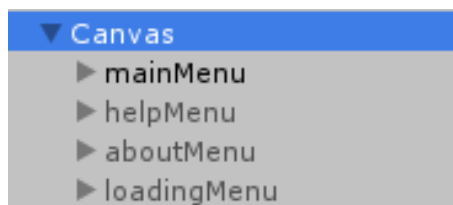


Figura 3.3 - Representação do objeto de jogo do tipo *Canvas* dentro da janela *Hierarchy*.

Os elementos filhos do painel *Canvas* são os quatro mostrados na Figura 3.3. O elemento *mainMenu* é o menu principal, o mostrado na Figura 3.2. Ele é o painel padrão ativo, ou seja, este será a tela mostrada quando a cena do menu principal for ativada, que acontece quando o jogo é inicializado ou quando é saído do ambiente de jogo.

O elemento *helpMenu* é uma tela que possui um pequeno texto explicativo, que auxiliará o profissional a se orientar dentro do jogo, mostrado na Figura 3.4.

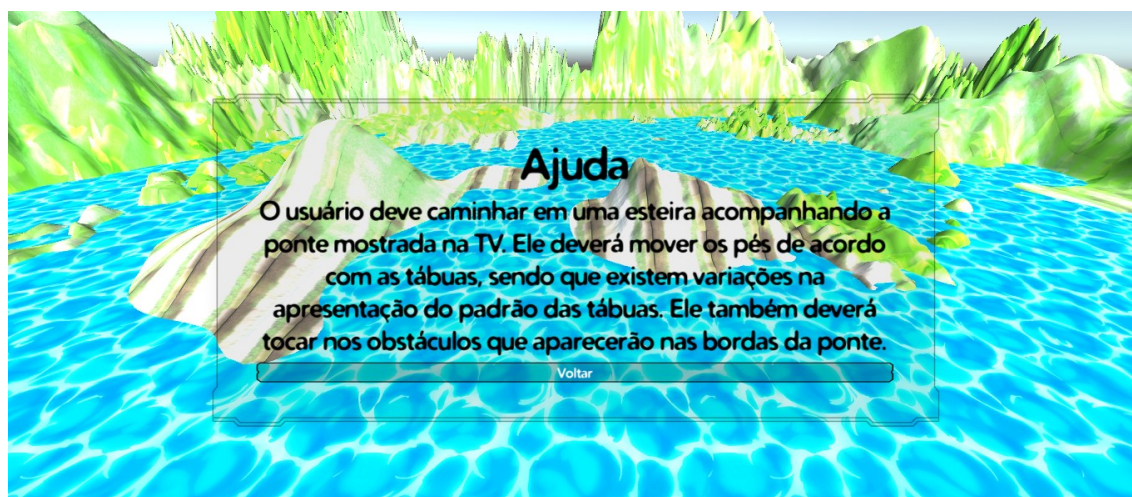


Figura 3.4 - Menu de ajuda ao usuário.

Já o elemento *aboutMenu* também é um pequeno texto, mostrado na Figura 3.5, que apenas mostra algumas informações do desenvolvimento do jogo.

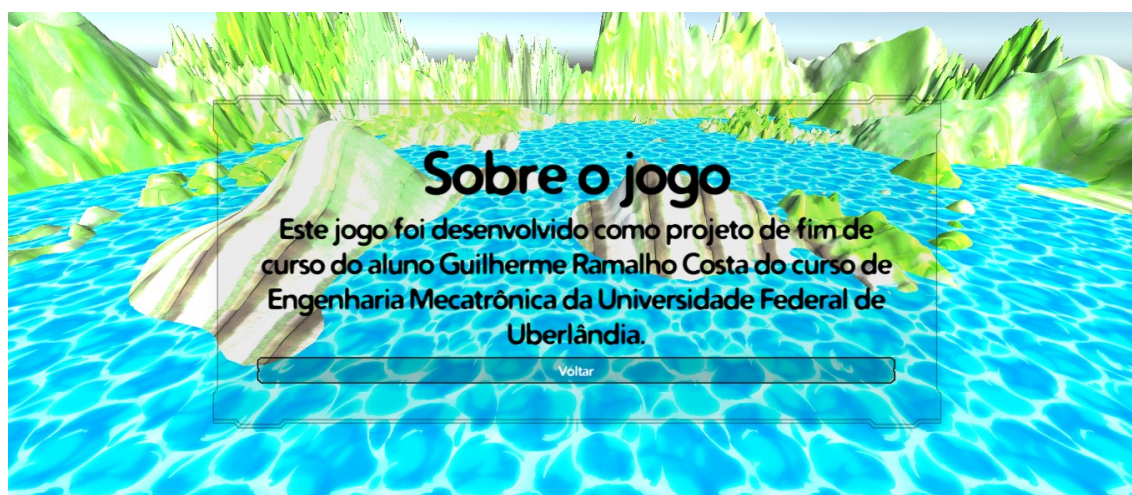


Figura 3.5 - Menu de informações sobre o jogo ao usuário.

Note também que há um elemento chamado *loadingMenu*. Esta tela é apenas uma forma de existir uma tela com informações sobre o carregamento de uma outra tela ou uma outra cena. Sendo mostrada na Figura 3.6, esta tela apenas aparecerá quando o jogo estiver carregando informações para serem apresentadas num momento posterior.



Figura 3.6 - Imagem da tela de carregamento entre cenas.

Nesta cena, temos os objetos mostrados na Figura 3.7. Todos estes são objetos de jogos, com componentes específicos para cada um deles.

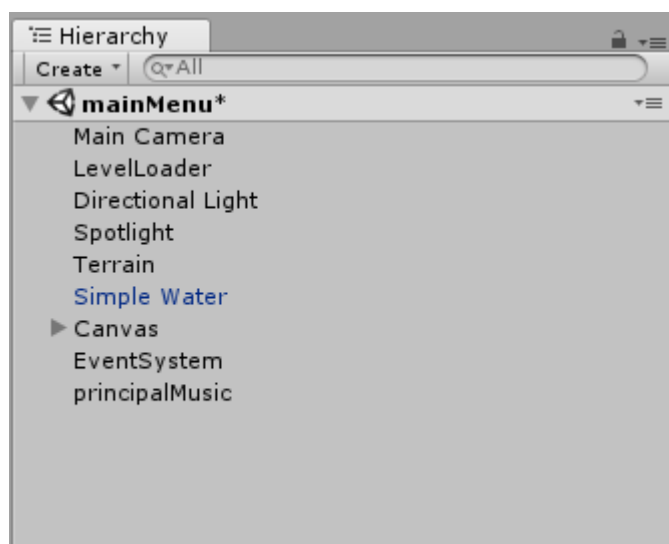


Figura 3.7 - A janela *Hierarchy* mostra os objetos de jogo da cena atual.

Será detalhado rapidamente cada um dos elementos mostrados na Figura 3.7. Como dito anteriormente, todos estes são objetos de jogo que possuem componentes que os caracterizam como este tipo de objeto.

O objeto *Main Camera* é a câmera virtual principal do jogo, que faz com que o usuário possa ver o cenário de fundo e o painel *Canvas*. Este objeto possui os componentes que o caracterizam como câmera e, como principal, possui o componente *Transform*, que foi modificado para que fosse mostrado a área de interesse do cenário.

Já o *LevelLoader* é o que se chama de objeto vazio. Este objeto vazio é como qualquer outro objeto de jogo, porém, fisicamente, ele não representa nada para o jogo. Utiliza-se este tipo de objeto para criar pontos de referência para a inserção de scripts relacionados a cena em questão.

Neste caso, o *LevelLoader* possui o script *Level Loader*, que é responsável por obter a porcentagem de carregamento para a próxima cena e, com estas informações, passar os valores necessários para a tela *loadingMenu*. Este script pode ser completamente visualizado no apêndice H.

Os objetos *Directional Light* e *Spotlight* são objetos de jogo responsáveis pela iluminação da cena. Estes possuem o componente *Light* que pode ser configurado de diversas maneiras para obter a intensidade, cor e outros aspectos de iluminação necessários para a cena em questão.

O objeto *Terrain* é o objeto de terreno do cenário. Ele é um objeto que foi totalmente desenvolvido pelas ferramentas gráficas do Unity 3D, não possuindo nenhum tipo de script escrito para controle.

O Unity 3D possui uma ferramenta de criação de terreno através de crescimento, aplainamento e decrescimento de um plano, utilizando texturas e pincéis variados, podendo serem padrões do Unity 3D ou importados de terceiros.

O objeto produzido para este projeto pode ser visto na Figura 3.8.

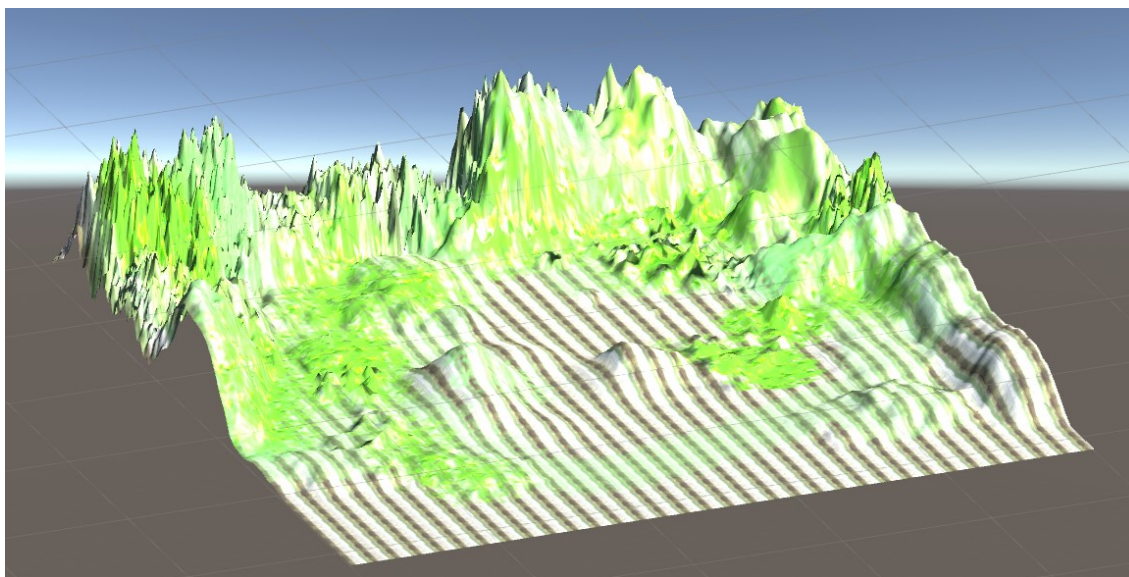


Figura 3.8 - Objeto de jogo que representa o terreno desenvolvido para o cenário.

Por sua vez, o objeto *Simple Water* foi um *asset* gratuito baixado pela loja do próprio motor de jogos Unity 3D. Este *asset* possui uma textura com movimentação, simulando a movimentação da água.

Ele foi inserido em um plano e, então, posicionado cruzando o objeto de jogo *Terrain*, simulando um lago.

De forma a melhorar a imersão do usuário ao jogo, foi desenvolvido um *script* de movimentação em outra direção neste plano, de forma que ele se movimenta para cima e para baixo, simulando a movimentação do lago causado pelo vento. Este *script*, que pode ser

visto no apêndice I, faz com que este plano alterne sua posição entre dois pontos ao longo de um tempo pré-definido no *script*.

O objeto *EventSystem* foi criado juntamente com os botões do elemento *Canvas*. Ele é necessário para que as ações que ocorrem ao se apertar os botões ocorram de forma correta e sem sobreposições.

Por fim, o elemento *principalMusic* é um elemento vazio, assim como o elemento *LevelLoader*. Ele apenas existe para que se possa utilizá-lo como objeto de jogo para inserir um componente de som que ficará reproduzindo ao longo de toda a cena. Neste caso, utilizou-se um som ambiente de ondas batendo na costa. Isto fará com que a experiência do jogador dentro do jogo seja mais intuitiva e imersiva.

Os controles deste arquivo de som são feitos exatamente nas configurações de seu componente. Pontos como momento de início de execução, execução em laço, volume, entre outros, são um exemplo de configurações que podem ser acessadas neste componente.

3.2.2. Interface de inserção de paciente

Nesta cena, há um formulário para ser preenchido pelo profissional que irá gerar um arquivo de texto em uma pasta previamente escolhida pelo mesmo. Desta forma, terá um registro da partida que irá acontecer.

A Figura 3.9 mostra uma visão geral desta cena.

Informações do usuário:

Nome:

Gênero:

Idade:

Data de início do tratamento:

Terapia de número:

Data de hoje:

Distância entre obstáculos (entre 10 e 100):

Salvar em:

Figura 3.9 - Tela principal da cena de inserção de usuário.

Esta cena possui apenas uma tela. Devido a sua importância, a mesma foi criada como uma cena separada de forma a melhorar o controle de gravação dos dados no sistema.

Ao clicar no botão “Iniciar” da cena “Menu principal”, o jogo carrega esta cena. Assim, o usuário deverá preencher os dados do paciente através do teclado de um computador, uma vez que a plataforma do jogo ainda é para ser executado em computadores.

As informações solicitadas neste formulário são:

1. Nome;
2. Gênero;
3. Idade;
4. Data de início do tratamento;
5. Terapia de número;
6. Data de hoje;
7. Distância entre obstáculos (entre 10 e 100), em metros; e
8. Salvar em.

Estas informações são inseridas dentro de uma caixa de texto posicionadas como filhos de um painel *Canvas*, assim como acontece na cena “Menu Principal”, juntamente com um botão de salvar.

Estas informações são controladas pelo *script* “SalvarArquivo.sc”, que pode ser visto completo no apêndice M.

Neste, é feita a operação com arquivos através do *script*. Após declarar todas as variáveis, cria-se um método para salvar os dados em um arquivo. Neste método, é criado o caminho para salvar o arquivo através dos campos “Salvar em”, “Nome” e “Terapia de número”. Os nomes dos arquivos são padronizados como “Nome - #Número da Terapia”. E, seu conteúdo é mostrado na Figura 3.10.

```
Nome: Guilherme Ramalho Costa
Gênero: Masculino
Idade: 22
Data de início do tratamento: 19/01/2018
Terapia de número: 1
Data de hoje: 19/01/2018
Distância entre obstáculos (entre 10 e 100): 15
Tempo de duração da sessão atual: 00hrs3min13.1s
Pontuação da sessão atual: 405 pontos!
```

Figura 3.10 - Exemplo de relatório gerado após cada partida.

Assim como na cena anterior, para utilização deste *script* em toda a cena, foi criado um objeto vazio chamado *User*, de forma que ele fique ativo durante toda a cena, esperando apenas o usuário clicar no botão salvar.

Nesta cena, não foi adicionado nenhum tipo de proteção contra o usuário, no sentido de falta de preenchimento ou preenchimento incorreto. O que o jogo está programado para fazer quando o usuário não preenche ou não preenche corretamente este formulário é não criar um relatório e utilizar as configurações padrões para executar a próxima tela.

3.2.3. Interface principal do jogo

Nesta cena, temos o jogo em sua execução. É aqui que o paciente irá caminhar na esteira enquanto o avatar movimenta na ponte.

Esta é a cena mais elaborada do jogo, uma vez que necessita de grandes quantidades de processamento e de controle para que sua execução seja correta. Nesta seção vamos falar sobre a interface em geral e, nas seguintes, falaremos mais especificadamente de cada um dos elementos principais que compõem essa cena.

A Figura 3.11 mostra a tela padrão desta cena. Ela consiste no personagem controlável pelo jogador através do Kinect caminhando em uma ponte sobre o mesmo cenário das outras duas cenas descritas anteriormente. Na ponte, irão aparecer esferas que deverão ser tocadas pelo jogador.



Figura 3.11 - Tela principal da cena de jogo.

Também na Figura 3.11, na parte superior direita, é possível ver um painel *Canvas* com informações sobre o desempenho do usuário, além de um botão para voltar para o menu. Esta cena irá ser executada até que este botão seja pressionado.

Os objetos do cenário são os mesmos mostrados anteriormente, apenas copiados para esta cena. Desde os objetos de jogos até seus *scripts* de controle.

3.2.4. Avatar

O avatar é o personagem principal do jogo, sendo desenvolvido para simular os movimentos do usuário, caminhando na esteira a frente do sensor Kinect, para dentro do jogo. Este personagem é um modelo pré-fabricado (*Prefab*) disponibilizado gratuitamente na internet para download por meio de *assets*.

Ele representa um modelo articulado, respeitando as posições das articulações do corpo humano. No jogo, ele é um conjunto de objetos de jogo coordenados hierarquicamente

de forma a serem ligados pelas articulações. O modelo utilizado pode ser visualizado na Figura 3.12.



Figura 3.12 - Detalhe do modelo utilizado para o avatar.

A hierarquia dos objetos de jogo componentes do modelo pode ser vista na Figura 3.13.



Figura 3.13 - Hierarquia da construção do avatar.

Note que este é um modelo complexo e, devido a boa performance do mesmo, não é justificável a fabricação deste tipo modelo para usos como o desenvolvido neste projeto.

Para controle do mesmo, o Kinect utiliza os pontos das articulações entre os objetos para mapear o corpo humano. Estes pontos são mostrados na Figura 3.14.

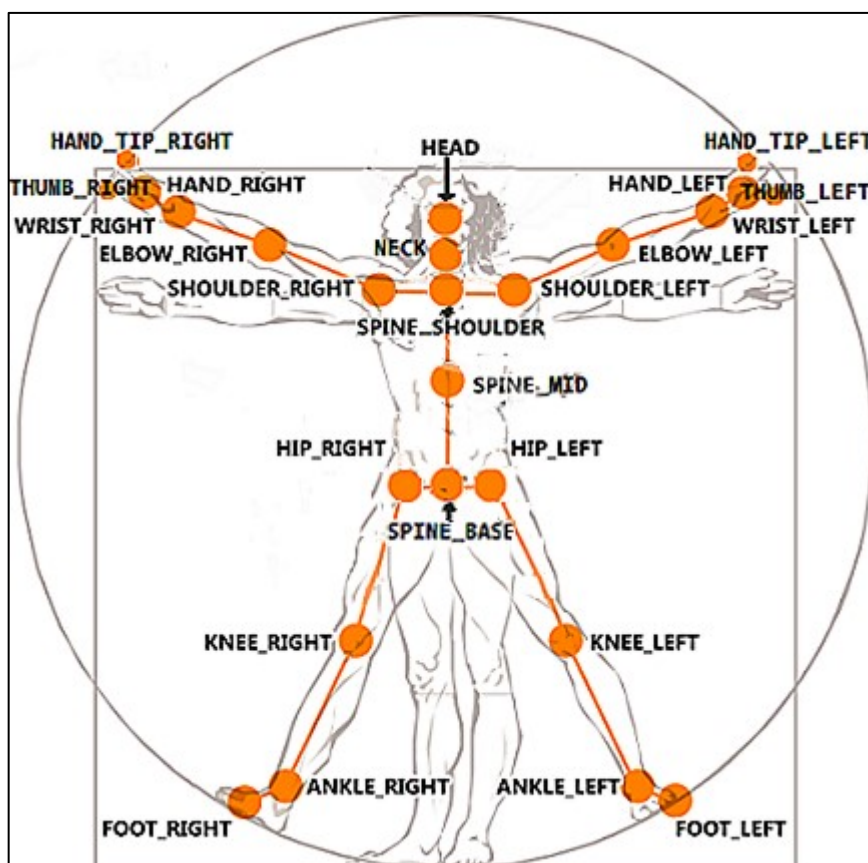


Figura 3.14 - Pontos identificados pelo sensor Kinect para a criação do esqueleto e das referências para o avatar.

Fonte: <https://goo.gl/eFuuT1> - acesso em 25/02/2018.

As articulações mostradas na Figura 3.14 são coincidentes com a relação dos objetos de jogo mostrados na Figura 3.13.

Uma vez que este personagem é um modelo pré-fabricado, seu controle pelo sensor Kinect é bastante facilitado. Utilizando o *package* mencionado anteriormente, é possível ter acesso ao conjunto de ferramentas de desenvolvimento que possui inúmeros *scripts* de demonstração e de controle.

Para controlar o avatar, utiliza-se o *script* “AvatarController.cs” utilizado como componente no objeto pai, mais alto da hierarquia, do personagem. Através do mesmo, é possível acessar as informações obtidas pelo sensor Kinect. Entretanto, não é apenas este *script* que é necessário. Para realizar o controle do Kinect também, deve-se criar um objeto de jogo vazio e, nele, adicionar outros três *scripts* como componentes: “KinectManager.cs”, “KinectGestures.cs” e “SimpleGestureListener.cs”.

Estes três *scripts* são necessários para o controle e obtenção de imagens a partir do sensor Kinect. O “KinectManager.cs” é responsável pelas características de posicionamento do sensor, máximo de usuários detectável, limitação de espaço de visualização entre outras informações. Ou seja, este *script* prepara o sensor Kinect para ser utilizado pelo jogo.

Por sua vez, o “KinectGestures.cs” é responsável por mapear os pontos do esqueleto e salvá-los em variáveis definidas de forma que os outros scripts possam acessar e reproduzir no avatar em questão.

Por fim, o “SimpleGestureListener.cs” é responsável por identificar os gestos e então repassar para o sistema. Neste *script* será diferenciado usuários quando existir 2 usuários na câmera, fazendo com que um não atrapalhe o outro. Embora o jogo seja feito apenas para 1 usuário por vez, foi pensado em deixar livre para o sensor de forma a não limitar sua atuação e permitir que o mesmo seja posicionado sem tanta rigorosidade.

Entretanto, ao deixar livre a identificação de usuários, faz-se a necessidade de apenas o paciente estar em frente ao sensor durante a execução do jogo. Assim, será necessário o desenvolvimento de um mecanismo que suporte e mantenha o paciente de pé para jogar ou o jogo deverá ser limitado apenas para aqueles pacientes que possuem um melhor equilíbrio e conseguem se manter sozinhos. O desenvolvimento deste suporte não é parte do desenvolvimento deste jogo.

3.2.5. Ponte

A ponte é um elemento de suma importância nesta cena. Ela é responsável pelo suporte do avatar, de forma que o mesmo possua uma superfície para caminhar e, desta forma, não caia.

A Figura 3.15 mostra este elemento em destaque em relação ao cenário. Note que ela não ultrapassa o cenário completo e sim apenas uma parte dele. Esta utilização do elemento para a sensação de caminho infinito é explicada melhor posteriormente.



Figura 3.15 - Visão da ponte dentro do jogo.

Este modelo foi desenvolvido através do software SolidWorks devido a facilidade e conhecimento prévio do autor com o mesmo e, também, por não ter disponível um modelo pré-fabricado, como o avatar, gratuito que atendesse as necessidades deste projeto.

Para a importação entre os dois softwares é necessário um terceiro, que transforma o arquivo 3D do objeto desenvolvido no SolidWorks em um modelo de objeto, para que o mesmo então possa ser utilizado como objeto de jogo. O procedimento de conversão entre um objeto no formato “.stl” para um objeto no formato “.obj” está descrito no apêndice R.

As texturas e cores também foram inseridas pelo software intermediário, uma vez que, para utilização no jogo, obteve melhores resultados utilizando texturas no formato aceito pelo jogo do que no formato desenvolvido e então importado posteriormente.

A ponte é um objeto de jogo que movimenta contra o usuário, dando a sensação de que o mesmo está caminhando ao longo daquela. Para que isto ocorra, utilizou-se do componente *Mesh Collider*, que faz uma malha da forma do objeto para que o mesmo seja passível de colisão, ou seja, quando os pés do avatar tocam a malha, eles não se sobrepõem.

Outros objetos que fazem parte desta ponte são dois pontos de *Trigger* (ou gatilhos). Eles são utilizados para sinalizar a criação de uma nova ponte e a destruição de uma ponte anterior.

O gatilho utilizado para a criação de uma nova ponte está posicionado aproximadamente na metade de cada segmento de ponte. Através do *script* “CriaPonte.cs” no apêndice E, quando o avatar passa por este gatilho, é criado um novo segmento de ponte logo a frente, em uma posição previamente específica.

Já o gatilho utilizado para a destruição de uma ponte antiga está posicionado ao final de cada segmento de ponte. Através do *script* “DestroiPonte.cs” no apêndice G, quando o avatar passa por este gatilho, o objeto de jogo é destruído de forma que não necessite mais de processamento. Para evitar que o avatar destrua uma ponte em que ele ainda esteja caminhando sobre a parte final do segmento sobrepõe a parte inicial do seguinte segmento por alguns pontos necessários para garantir que o avatar já caminha sobre o novo segmento ao destruir o anterior.

3.2.6. Obstáculos

Os obstáculos foram desenvolvidos de forma a motivarem o movimento dos braços durante a caminhada, fazendo com que o paciente execute uma ação fora do padrão da marcha, fazendo com que a mesma se torne mais real e possa treinar o balanço do corpo necessário para a marcha e reabilitação desta.

Através do obstáculo mostrado na Figura 3.16, o paciente poderá desenvolver habilidades de executar não só a marcha, mas também de melhorar sua atenção durante a execução da mesma.

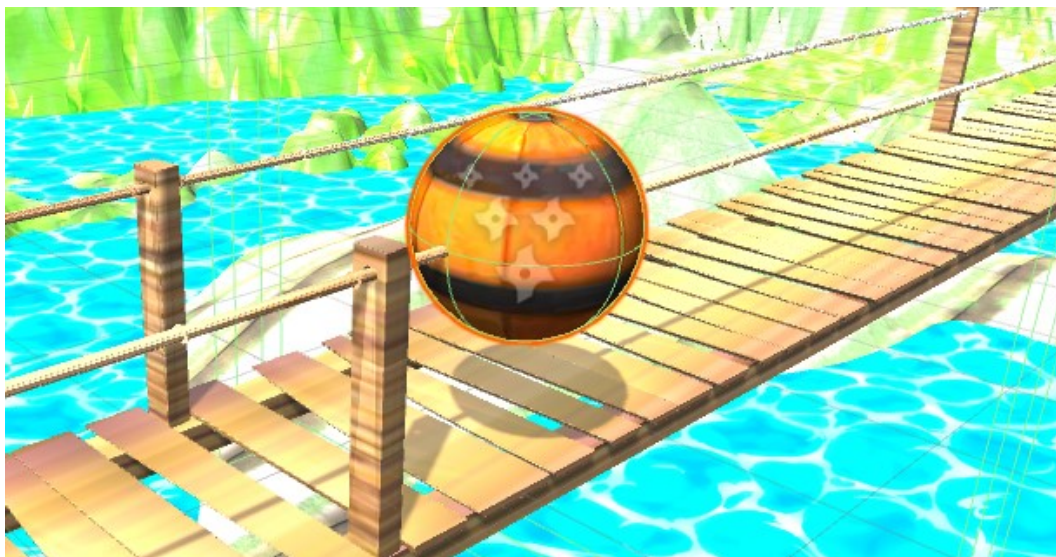


Figura 3.16 - Visão em detalhe do obstáculo dentro do jogo.

Os obstáculos estão sempre posicionados lateralmente na ponte, de forma que sejam facilmente tocados pelo usuário. Quando tocados, os mesmos produzem um efeito, se transformando em uma explosão de estrelas, apenas para cunho de entretenimento. Esta forma do obstáculo é mostrada na Figura 3.17.



Figura 3.17 - Efeito de explosão de estrelas que acontece quando o usuário toca nos obstáculos.

O posicionamento dos obstáculos é feito pelo *script* “PosicaoAleatoriaObs.cs”, presente no apêndice J. Este *script* verifica o valor dado pela distância entre os obstáculos na cena anterior, de inserção de paciente e, através deste valor, cria um vetor com duas

posições. A primeira posição é para quando os obstáculos estão do lado direito do avatar e, a segunda, do lado esquerdo.

Então, toda vez que o avatar passa por um obstáculo, é gerado um número randômico inteiro, zero ou um. Desta forma, caso seja zero, o obstáculo irá ser posicionado do lado direito e, caso seja um, do lado esquerdo do avatar.

Para executar o efeito de explosão em estrelas, utilizou-se o sistema de partículas do motor de jogos Unity 3D. Este sistema permite a criação de partículas que podem ou não se mover de formas determinadas. É bastante utilizado em explosões e em obstáculos aleatórios que não possuem uma regra (diferentemente dos obstáculos neste caso).

Para que as partículas fossem iniciadas, utilizou-se a mesma técnica de gatilho. Através do *script* “PosicaoAleatoriaObs.cs”, o mesmo para gerar novos obstáculos, quando o avatar colide com o obstáculo, ele inicia a explosão de partículas.

Assim como na ponte, é necessário destruir os obstáculos que já passam pelo avatar, para que não haja a necessidade de processamento de objetos de jogo que não estão aparecendo na cena.

Desta forma, também no *script* “PosicaoAleatoriaObs.cs” há a adição de um novo componente no objeto pai, que é o objeto de jogo “Obstáculo 1”. Este componente é o *script* “DestroiObstaculo.cs”, que inicialmente, faz com que o objeto de jogo em questão fique desativado e, após um tempo pré-determinado, é destruído para economia de processamento.

3.2.7. Sistema de pontuação e tempo de partida

Este sistema é um painel *Canvas* que está no campo superior direito da cena do jogo. Diferentemente das outras cenas, este é o único painel *Canvas* que não ocupa toda a tela em questão.

A Figura 3.18 mostra este painel. Ele é composto por informações de pontuação, tempo de partida e um botão para retornar ao menu principal.

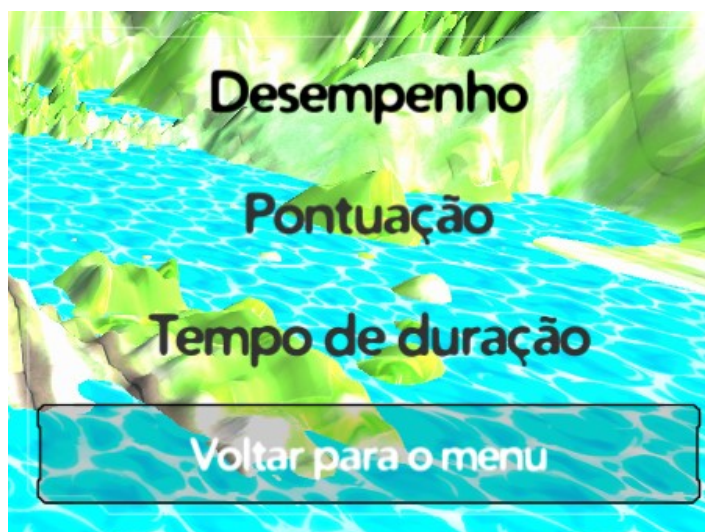


Figura 3.18 - Painel *Canvas* de informação de desempenho do jogador.

Ele obedece às informações e técnicas utilizadas nos painéis *Canvas* das cenas 1 e 2. Desta forma, não serão repetidos estes conceitos, apenas será discutido sobre a forma de amostragem para o tempo e para a pontuação.

A pontuação é proporcional a velocidade do avatar. Isto foi definido de forma que, quando o paciente estiver apto a caminhar a maiores velocidades, a pontuação do mesmo aumente, fazendo-se possível um acompanhamento da evolução do mesmo.

O *script* “VariacaoVelocidadeAvatar.cs”, apresentado no apêndice P, utiliza a velocidade do avatar, para incrementar os pontos a cada 25 frames. Este intervalo de frames é utilizado para que a pontuação não cresça incessantemente durante a partida e alcance resultados gigantescos.

Este mesmo *script* limita que a pontuação cresça apenas se a velocidade do avatar seja maior do que 0,01 m/s. Isto faz com que pequenos movimentos não planejados não sejam captados pelo sistema de pontuação.

O sistema de tempo de duração é utilizado com base no sistema de temporização do próprio Unity 3D, da classe *Time*. Ele obtém, a cada frame, o tempo desde o carregamento da cena atual, em segundos.

O *script* “TempoExecucao.cs”, no apêndice O, faz com que este tempo obtido a cada frame seja convertido em uma *string* no formato `##hrs ##min ##.##s` para que fique melhor compreensível para o usuário do tempo corrido durante a partida.

Os valores de pontuação e de tempo são salvados no mesmo arquivo de relatório gerado na cena 2, a partir do *script* “salvarTempoEPontos.cs”, no apêndice N.

Este *script* obtém os valores no painel *Canvas* e, é acionado ao clique do botão presente no mesmo. Após o clique, ele abre o arquivo existente a partir do nome do arquivo de relatório salvo na cena 2, passado por meio da classe *PlayerPrefs*, que consegue transmitir informações entre cenas do jogo.

Portanto, as duas últimas linhas da Figura 3.10 são adicionadas apenas quando o terapeuta clica no botão “Voltar para o menu” na cena 3.

3.3. Métodos de controle

Nesta seção, será tratado mais especificadamente dos métodos de controle utilizados nas principais etapas do jogo. Aqui serão mencionados o controle entre cenas, controle da velocidade do avatar, alterações de cenário entre outras informações.

3.3.1. Menus de utilização

O principal mecanismo de controle nos menus é a existência de controle dos eventos entre as ações dos botões e preenchimentos de caixas de texto. Neste caso, irá se detalhar a forma utilizada de controle nestas ações.

Nos objetos de jogo que representam os botões, temos um componente chamado *Button (Script)* que já é carregado para o mesmo no momento da inserção do botão no painel *Canvas*. Neste componente, temos uma configuração chamada *On Click ()* que possui, de forma facilitada, a organização das ações realizadas ao clicar de um botão.

A Figura 3.19 mostra esta parte do componente em questão.

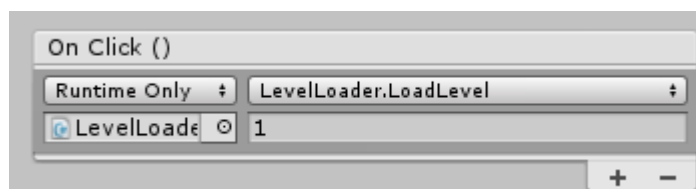


Figura 3.19 - Configuração "On Click ()" dentro do componente "Button (Script)".

A partir deste conceito de ação ao clicar, desenvolveu-se *scripts* de alternância entre cenas, telas e de saída do jogo. Vamos falar um pouco de cada um deles na sequência.

O *script* “LevelLoader.cs” já foi mencionado anteriormente. Quando dentro do componente dos botões, é possível selecionar um método dentro do *script*, neste caso, o método *LoadLevel (int sceneIndex)*, ele fica esperando a ação de clicar para ser ativado e, dentro do componente, o parâmetro *sceneIndex* pode ser passado. Este parâmetro representa a hierarquia da cena em questão a ser carregada ao clique do botão.

Para gerar a hierarquia dentre as cenas do mesmo projeto, deve-se inclui-las nas configurações de construção do jogo. Assim, é aberta uma nova janela onde deve-se incluir as cenas do jogo e, então, elas serão classificadas com valores inteiros para que possam ser referenciadas de forma fácil pelos *scripts* e configurações dentro do Unity 3D.

A Figura 3.20 mostra a janela para organização das cenas na construção do jogo.

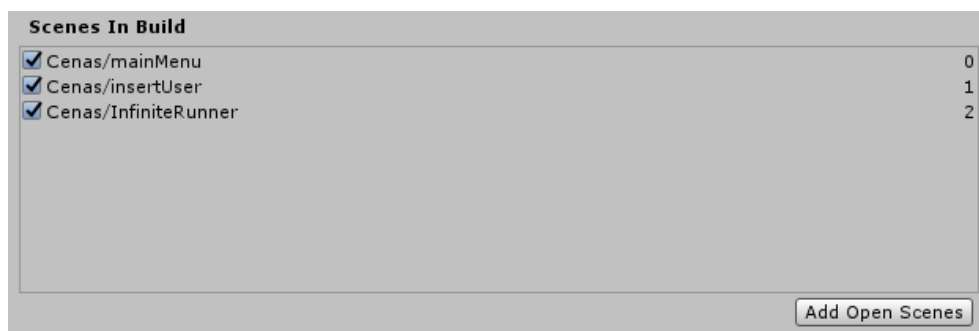


Figura 3.20 - Caixa de seleção das cenas que fazem parte do jogo em construção.

A ação responsável por alternar telas em uma mesma cena não necessita de um script específico para tal. Utilizando o conceito de objeto de jogo, o organizador de eventos *On Click* () possui ações pré-estabelecidas para objetos de jogos. Uma delas é a ativação e desativação destes objetos. Como a alternância entre telas nada mais é do que isto, utilizou-se deste procedimento para alternar entre as telas de “Menu Principal”, “Ajuda” e “Sobre” na cena em questão. Ao se clicar no botão “Ajuda”, por exemplo, o que acontece é a desativação do painel do menu principal e a ativação do painel do menu ajuda. Sem a necessidade de um *script* específico para tal ação.

Por fim, a realização de sair do jogo com o clique de um botão foi gerada a partir de um *script*, o “Sair.cs”, no apêndice L, é responsável por para a execução da aplicação. Assim como é possível acessar a um método pelo componente, acessou-se o método *sairJogo* (), que finaliza a execução.

3.3.2. *Marcha do avatar*

De forma a sincronizar a velocidade do avatar com a velocidade da marcha do paciente jogador, utilizou-se algumas informações de juntas específicas do avatar para servirem como referência do cálculo da velocidade.

Durante uma caminhada na esteira, pode-se notar que os únicos membros que estão em movimentação periódica são os membros inferiores. Isso acontece, pois, a velocidade é ditada pelo ritmo selecionado na esteira e não pelo usuário.

Desta forma, escolheu-se como referência uma junta no pé do avatar, de forma que se garantiu que a movimentação seria o mais constante possível.

A Figura 3.21 mostra um exemplo de identificação do esqueleto pelo sensor Kinect e, as duas setas identificam as juntas utilizadas para a determinação da velocidade na esteira.

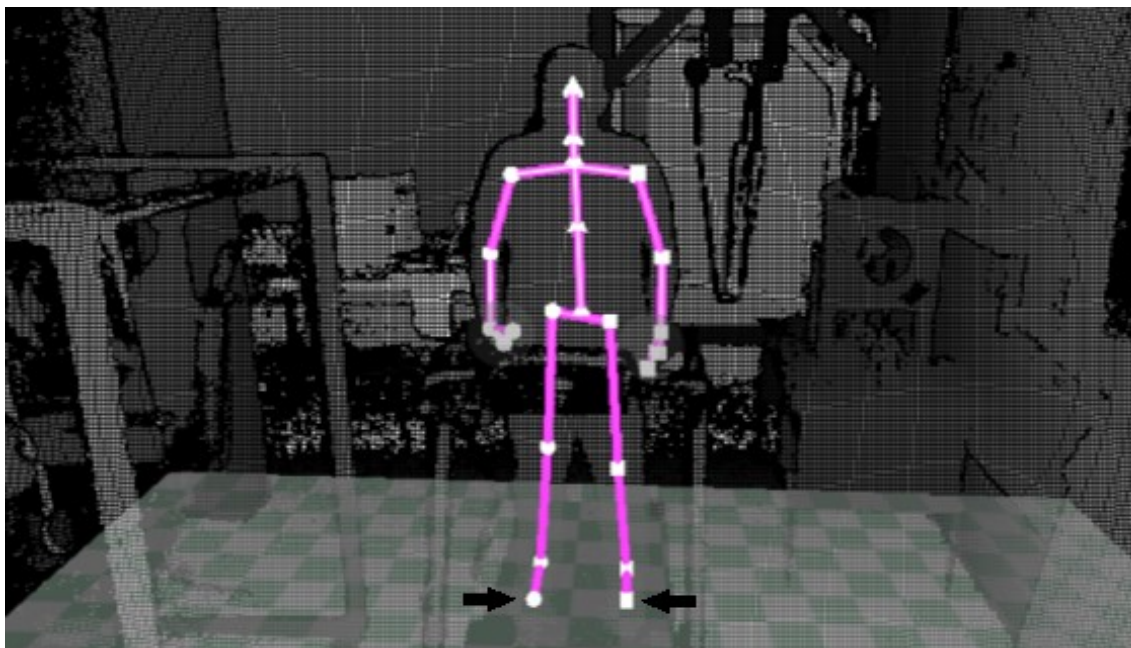


Figura 3.21 - Imagem gerada pelo sensor Kinect para identificação do corpo humano e alguns pontos de articulação.

Uma vez identificado esta junta, tem-se as informações da posição (x,y,z) da mesma dentro do jogo a cada frame. Assim, pode-se obter estes valores em uma determinada variação de tempo. Para a escolha deste valor de tempo, atentou-se ao fato de que alguns valores poderiam coincidir com uma passada completa, dando a sensação para o jogo de que a junta quase não se movimentou.

Desta forma, empiricamente, foram testados vários valores até atingir uma velocidade alta na esteira para os padrões de caminhada, 5 km/h, onde o avatar ainda estava apto a acompanhar o usuário e, também, gerar a mesma velocidade do usuário. Desta forma, o intervalo de tempo entre a aquisição destes valores ficou como sendo de 0,05 segundos.

Para que esta aquisição não seja refém da taxa de variação de frames, não se utilizou o método Update (). Porém, para que o jogo produza a sensação conhecida como *Infinite Runner*, onde o avatar percorre um caminho sem vislumbre do final deste, utilizou-se a movimentação da ponte para gerar tal sensação.

Já que o avatar é um modelo completo, com processamento alto e com muitos objetos, optou-se por não adicionar mais um componente ao objeto do mesmo, a velocidade. Assim, durante o jogo, mesmo com a sensação de movimentação, o avatar sempre está parado e é a ponte de se movimenta em direção oposta ao sentido da marcha do usuário.

Devido ao fato de que a velocidade está sendo gerada em um objeto e aplicada em outro, notou-se um atraso na resposta desta velocidade no jogo. Para consertar tal revés, utilizou-se um fator de correção, aumentando em 60% a velocidade entregue para o objeto de jogo "Ponte".

Os *scripts* responsáveis pela obtenção de todas as informações apresentadas são o “VelocidadePonte.cs” e o *script* “VariacaoVelocidadeAvatar.cs”, apêndice Q e P, respectivamente. O primeiro tem como função a identificação dos pontos necessários para a realização do cálculo de velocidade, neste caso, utilizou-se a média das juntas do pé direito e do esquerdo, além de repassar este valor de velocidade para o objeto de jogo em que está adicionado.

Já o segundo, tem como função o real cálculo da velocidade. Pegando os valores das coordenadas das duas juntas em dois momentos no tempo e retornando o valor da velocidade dentro deste tempo.

Os objetos que se movimentam no jogo possuem o *script* “VelocidadePonte.cs” em seus componentes, por exemplo, a ponte e os obstáculos. Já os objetos que se movimentam na realidade, os pés, possuem o *script* “VariacaoVelocidadeAvatar.cs” nos seus componentes.

3.3.3. Controle de alterações

No decorrer do jogo temos alterações que devem ser controladas pelo ritmo de marcha do usuário.

Uma vez que o jogo é uma corrida infinita, tem-se a repetição de padrões a todo momento. Por um lado, torna-se fácil a inserção de novos objetos e exclusão dos antigos. Por outro, tem-se a necessidade de identificação de posições onde os objetos antigos estavam para que continue sem que o usuário perceba esta mudança.

Assim, como dito no tópico anterior, o controle da velocidade passa para os objetos do jogo em movimento, a ponte e os obstáculos. Com a necessidade de serem contínuos, existe o controle de criação e destruição destes objetos.

A ponte é controlada por meio de gatilhos de criação e destruição da mesma, posicionados estrategicamente no decorrer dela. Estes gatilhos são objetos filhos do objeto de jogo “Ponte” e, dessa forma, não necessitam de possuir o componente de velocidade, apenas o objeto pai.

Um gatilho é um objeto que se mostra invisível para o usuário. Pode ter formas variadas e, geralmente, são utilizados para acionar ações automáticas uma vez alcançados certa configuração dentro do jogo. Neste caso, esta configuração é uma parte específica da ponte.

A Figura 3.22 mostra a posição do gatilho no decorrer da ponte. Tendo o modo de visualização da cena modificado para visualização das malhas, é possível observar o gatilho como um paralelepípedo em verde.

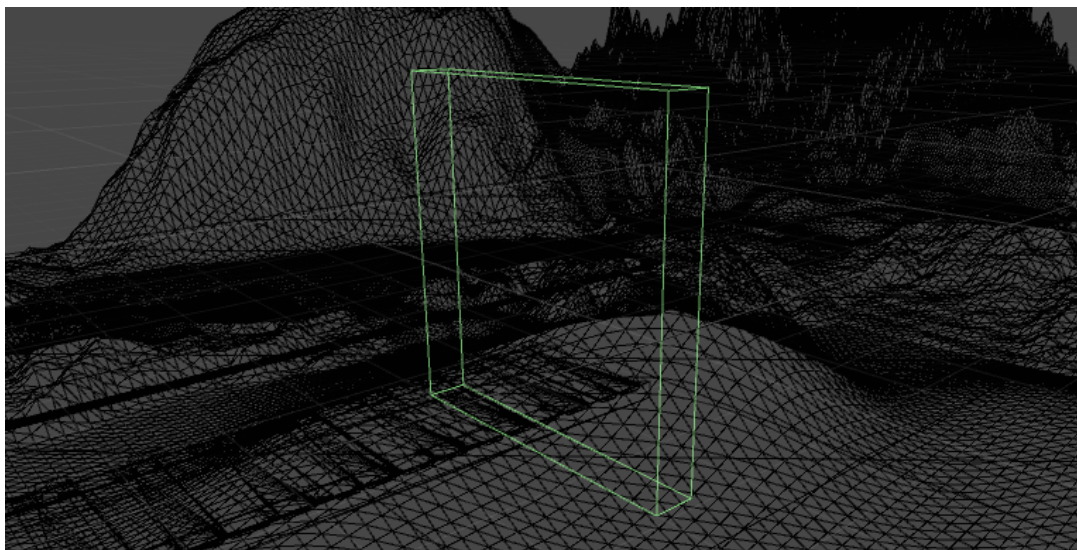


Figura 3.22 - Representação do gatilho (em verde) dentro do mundo do jogo (em preto).

Uma vez acessado o gatilho de criação, este posiciona um novo segmento de ponte através de um modelo idêntico ao atual salvo como um modelo pré-fabricado (*Prefab*). Desta forma, tem-se a garantia de que o novo objeto posicionado é um clone do objeto anterior, possuindo todos seus componentes idênticos, tendo apenas seu componente *Transform* modificado pelo *script* “CriaPonte.cs”.

Os obstáculos possuem um pouco mais de complexidade em seu posicionamento contínuo. Uma vez que existe mais de um obstáculo por segmento de ponte que surge ao longo da movimentação, este não pode se comportar como um objeto de jogo filho do objeto “Ponte”.

Assim, criou-se um objeto de jogo “Obstáculo” que possua o componente “VelocidadePonte.cs”, de forma que a velocidade relativa entre o objeto de jogo “Obstáculo” e o objeto de jogo “Ponte” seja nula. Desta forma, resolvemos o problema de movimentação do obstáculo em uma mesma posição sem ser um objeto de jogo filho do objeto “Ponte”.

Como citado anteriormente, há duas posições possíveis para o posicionamento dos obstáculos, uma do lado direito e outra do lado esquerdo. Os obstáculos aparecem aleatoriamente ao longo da ponte no momento em que o obstáculo anterior é tocado pelo avatar, realizando a explosão de estrelas.

Para que isto seja possível, criou-se dois objetos de jogo pré-fabricados para os obstáculos, o objeto “Obstáculo 1” e o objeto “Obstáculo 2”. Desta forma, ao se tocar em um obstáculo, há um gatilho que cria um clone do outro obstáculo, na posição selecionada no *script* “PosicaoAleatoriaObs.cs” e assim sucessivamente, sempre alternando na criação dos obstáculos.

Isto também ocorre para que, quando o obstáculo anterior for destruído, o próximo obstáculo não seja também, uma vez que, ao se criar um clone, ele acabará se vinculando ao

seu antecessor. Ao se criar objetos alternados, na prática, não estarão sendo criados clones e sim novos objetos idênticos aos anteriores e sem ligação entre os mesmos.

3.4. Testes de experiência – *Game Experience Questionnaire (GEQ)*

Nesta seção será descrito sobre a utilização do GEQ para mensurar melhor o impacto do jogo em seus usuários. Desenvolvido e validado pelo projeto FUGA, o GEQ trata de forma confiável sete dimensões diferentes na experiência de um jogador: Imersão Sensorial e Imaginativa, Tensão, Competência, Fluidez, Efeito Negativo, Efeito Positivo e Desafio (APPL et al., 2008; IJSSELSTSIJN et al., 2008).

Há diversos trabalhos na literatura que utilizaram este tipo de qualificação para mensurar a imersão da experiência de jogadores (NACKE et al., 2009; DRACHEN et al., 2010; CHRISTOU; MICHAEL, 2014; MACHADO, 2015).

Existem variações do questionário para diversas aplicações. Para este trabalho em questão, utilizou-se uma redução do questionário original, que contém 42 questões divididas nas sete dimensões da experiência do jogador. Na fase inicial deste projeto, utilizou-se apenas uma questão que resume, em geral, a experiência em cada uma das dimensões.

A Tabela 3.1 mostra quais foram as afirmações utilizadas e em qual dimensão cada uma delas atua.

Tabela 3.1 - Relação entre as afirmações utilizadas no questionário e as dimensões avaliadas.

	Afirmativa	Dimensão
1	O jogo era esteticamente agradável	Imersão Sensorial e Imaginativa
2	Eu me senti frustrado	Tensão
3	Eu me senti habilidoso	Competência
4	Eu me senti imerso	Fluidez
5	Eu o achei cansativo	Efeito Negativo
6	Eu o achei divertido	Efeito Positivo
7	Eu me senti desafiado	Desafio

Para quantificar estas afirmativas, escolheu-se utilizar a escala de Likert, uma escala que possui uma quantificação de 5 pontos, muito utilizada no contexto de ciências comportamentais. Ela consiste em, a partir de uma afirmação, os respondentes irão emitir seu grau de concordância (SILVA JÚNIOR; COSTA, 2014).

De forma a melhor atender as informações solicitadas, a escala foi graduada da seguinte forma:

- 1 – De modo algum;

- 2 – Levemente;
- 3 – Moderadamente;
- 4 – Bastante; e
- 5 – Extremamente.

Com esta graduação, é possível determinar melhor o grau de impacto positivo e negativo nos usuários durante as partidas do jogo.

3.4.1. Usuários

Os usuários participantes do teste foram amostrados aleatoriamente dentro da Universidade Federal de Uberlândia (UFU) e totalizaram 10 respondentes.

Os testes mostrados neste documento foram feitos com usuários saudáveis em relação a marcha humana. Desta forma, garantiu-se que todos os respondentes do teste estavam aptos a participarem dos testes sem quaisquer ônus a saúde dos mesmos.

Deve-se atentar ao fato de que testes ainda devem ser realizados com uma amostra de usuários que atendam aos requisitos terapêuticos deste projeto, ou seja, pacientes pós-AVE que possuam padrão anormal da marcha. Estes deverão ser avaliados por um profissional capacitado e, só então, deverão ser organizados para participarem do teste. Desta forma, o teste realizado neste projeto atesta apenas o caráter lúdico e sensorial do mesmo em pessoas saudáveis. Ao ser utilizado em seu público alvo, pacientes em processo de reabilitação, os dados mostrados neste projeto poderão ser alterados. Este processo de análise em pacientes está sugerido como um projeto posterior.

3.4.2. Procedimento para os testes

Com o intuito de padronizar os testes ao máximo, definiu-se características que pudessem ser replicadas em todo o procedimento de teste.

Inicialmente, explicou-se para o usuário o objeto de estudo e o objetivo do mesmo. Fez-se um apanhado geral sobre o desenvolvimento do jogo e como o mesmo atua.

Todos os testes foram realizados durante 1 minuto e 30 segundos, na velocidade da esteira escolhida a 1 km/h. O posicionado dos obstáculos foi escolhido para uma distância a cada 15 metros, distância suficiente para que surgissem mais de um obstáculo durante o trajeto.

Solicitou-se que os usuários estivessem calçados com sapatos fechados ou, caso não fosse possível, que estivessem descalços para evitar movimentação incorreta dos pés em sapatos que não são presos aos mesmos, como ocorre em chinelos e sandálias.

Após a partida, orientou-se o usuário a responder o GEQ através de um formulário feito na plataforma Forms® da empresa Google, contendo as sete afirmações da Tabela 3.1 e uma escala Likert para cada uma das respostas. Todas as respostas são sigilosas a fim de evitar respostas tendenciosas positivas ou negativas. O formulário está apresentado no apêndice S.

CAPÍTULO IV

RESULTADOS E DISCUSSÕES

Os resultados do jogo, enquanto software, se mostraram satisfatórios, já que atenderam a todos os requisitos inicialmente levantados. A utilização do motor de jogos Unity 3D facilitou o desenvolvimento de um jogo esteticamente agradável e que não perdesse a essência da seriedade da reabilitação.

O apêndice A mostra os passos da criação do jogo completo. Nele é possível observar a utilização das ferramentas de auxílio dentro do motor de jogos ao invés de programação pura para controle de objetos. Também é possível notar como os *scripts* são simples e intuitivos, uma vez que os objetos são de fácil referência dentro do jogo.

Entretanto, em momentos onde o motor de jogos Unity 3D insiste em auxiliar em procedimentos, acaba-se por limitar certas ações ou torna-las difíceis de serem implementadas. Isto ocorre com a utilização dos gatilhos juntamente as colisões.

Ao facilitar o acontecimento destas colisões gerando gatilhos em objetos transparentes ou não, o motor de jogos Unity 3D pressupõe que as colisões serão continuadas e não apenas uma. Isto fez com que o objeto de jogo “Avatar” tivesse que ser modificado do padrão, perdendo seus contatos de colisão nos pés e nas mãos e transformados em apenas uma grande cápsula. Isto fazia com que a colisão acontecesse apenas uma vez, por mais tempo que o avatar ficasse no mesmo local.

Em relação aos aspectos quantificados durante os testes de experiência no jogo, tem-se a Figura 4.1.



Figura 4.1 - Gráfico representativo das respostas do teste de experiência de jogo.

Durante os testes, pôde-se observar a eficácia do mesmo em definir as opiniões dos respondentes em relação ao jogo, uma vez que os mesmos se mostraram tranquilos em relação as afirmações e aos graus de concordâncias da escala de Likert utilizados.

Para uma melhor análise dos dados gerados pela escala, utilizou-se da moda dos valores para cada uma das afirmações. Ao se utilizar a moda, tem-se as informações sobre o grau de concordância mais frequente sobre cada afirmação. Para este tipo de caso, utilizar uma medida de tendência central como a moda representa melhor o contexto, devido a pequena amplitude dos intervalos. Para identificar a moda na Figura 4.1, deve-se procurar pela coluna mais alta de cada uma das sete afirmações.

Para a afirmação “Eu me senti desafiado.” temos 6 respondentes que escolheram o grau de concordância “Bastante”. Pode-se concluir duas ideias desta resposta: os respondentes se imaginaram como pessoas em reabilitação e, devido a este fato, imaginaram-se desafiados a voltar a caminhar. Ou os respondentes se sentiram desafiados uma vez que a velocidade de marcha utilizado, 1 km/h, é incomum para pessoas sem problemas na marcha, ou seja, saudáveis. Além do fato de que todos tiveram o primeiro contato com o jogo no momento do teste e, também, podem não estar familiarizados com os equipamentos utilizados (como uma esteira).

Já para a segunda afirmação, “Eu me senti habilidoso.”, temos um total de 5 respondentes para os graus de concordância “Moderadamente” e “Bastante”, configurando um empate para a determinação da moda. Conclui-se que, neste caso, os respondentes

imaginaram-se na aplicação deste jogo sério para a reabilitação da marcha humana e em como o jogo poderia trazer um sentimento de aptidão para os usuários.

Por sua vez, a terceira afirmação, “Eu me senti imerso.”, trouxe como moda o grau de concordância “Extremamente”. Possivelmente, este resultado se deu devido aos aspectos gráficos do jogo. A inserção da movimentação da água, juntamente com o posicionamento da câmera como um objeto filho do objeto de jogo “Avatar” trouxe movimentos bastante similares aos reais, fazendo com que existisse uma sensação de imersão dentro do jogo.

Para a afirmação “Eu o achei cansativo.”, temos uma tendência central de 7 respondentes no grau de concordância “De modo algum”. Esta afirmação julga, além do cansaço físico, o cansaço visual e psicológico, ou seja, se existe informações exageradas durante a partida, fazendo com que o usuário chegue a ficar cansado.

Para a afirmação “Eu o achei divertido.” temos o valor da moda como “Extremamente”. Possivelmente, este valor foi obtido uma vez que os respondentes imaginaram o impacto deste tipo de jogo em reabilitações onde, tradicionalmente, os exercícios são monótonos e cansativos.

Na afirmação que julga a imersão sensorial e significativa, “O jogo era esteticamente agradável.” temos o resultado de “Extremamente” com 8 respondentes. Isto é um reflexo das afirmações “Eu o achei cansativo.”, “Eu me senti imerso.” e, também, da afirmação “Eu o achei divertido.” pois, através destas informações, espera-se que o jogo atendesse aos critérios de beleza e agradabilidade para cada um dos participantes do teste.

A afirmativa “Eu me senti frustrado.” traz como unanimidade o grau de concordância “De modo algum”. Isto mostra, a facilidade de desenrolar do jogo e a forma como o mesmo foi desenvolvido, sem condições de finalização de partida caso o usuário não consiga realizar as movimentações. Fazer com que o jogo não possua condições que definam quem ganha ou quem perde traz para o usuário a sensação de que, qualquer avanço realizado pelo mesmo, é digno de pontuação e, conseqüentemente, é uma vitória.

Por fim, deve-se atentar ao fato de que os resultados para uma melhor análise do jogo devem ser extraídos de uma amostra de jogadores que possuam um padrão anormal da marcha proveniente de um episódio de AVE.

Os testes que geraram as informações contidas na Figura 4.1 foram produzidos nas dependências do Laboratório de Automação e Robótica da Faculdade de Engenharia Mecânica da Universidade Federal de Uberlândia, utilizando seus equipamentos.

CAPÍTULO V

CONCLUSÕES

Com a realização deste projeto, foi possível desenvolver um jogo sério voltado para a reabilitação da marcha em pacientes pós-AVE e, desta forma, espera-se melhorar os processos de reabilitação para estes usuários.

A elaboração do planejamento do jogo sério foi uma das principais atividades, uma vez que, embora um instrumento de criação lúdica, o mesmo tem como objetivo final o auxílio na reabilitação dos pacientes.

A criação do jogo em si, auxiliado pelo motor de jogos Unity 3D juntamente com o uso do sensor Kinect, utilizando uma versão gratuita do programa e um pacote de desenvolvido do sensor, foi importante para o alcance do produto final. A facilidade de desenvolvimento e integração entre estas duas plataformas fez com que o jogo desenvolvido fosse altamente aceito entre os respondentes do teste de experiência.

O desenvolvimento de um cenário onde o usuário pudesse se sentir imerso dentro do jogo, utilizando características visuais e sonoras também resultou em um impacto positivo, de acordo com as respostas obtidas nos testes de experiência de jogo. A adaptação do avatar para reprodução das ações do usuário dentro do jogo também foi importante no desenvolvimento da sensação de imersão dentro do jogo.

O método definido para ligação entre a velocidade desenvolvida na esteira e a reproduzida pelo caminhar do avatar também trouxe bons resultados. Entretanto, alguns pontos se tornavam um pouco mal desenvolvidos devido a problemas de captura e de segmentação do próprio sensor Kinect. Porém, estes não acarretaram grandes problemas ou comprometeram o desenvolvimento e execução do jogo.

A utilização dos mecanismos de criação dos painéis de interface do usuário (UI) também trouxe grande facilidade na criação de menus e também da tela de inserção de

informações do usuário, juntamente com a tela de informações de desempenho junto ao jogo. A facilidade de criação destes se deve a boa arquitetura do motor de jogos utilizado.

A determinação da não existência de condições de parada, mas, também, a relação entre a velocidade do paciente, seu desempenho na marcha e sua pontuação fez com que os profissionais da área das terapias de reabilitação que acompanhassem o jogo pudessem observar melhora quantificada no desenvolvimento dos seus pacientes enquanto os mesmos não se sentiam frustrados com o jogo, caso perdessem ou não conseguissem executar determinadas tarefas.

E a realização dos testes, que fez com que a validação do jogo, em seu caráter lúdico e sensorial, acontecesse de forma positiva entre os respondentes.

Também, após o desenvolvimento do jogo, idealizou-se a realização de trabalhos posteriores, de forma a melhorar o projeto atual. Dentre estes trabalhos, pode-se citar a realização da validação do projeto junto a uma amostra de jogadores que possuam um padrão anormal da marcha após um episódio de AVE. Também será necessário o acréscimo de novos parâmetros terapêuticos a fim de tornar o jogo melhor aproveitado pelos profissionais da terapia. Outro ponto interessante é uma constante melhoria da interface do jogo, de forma ao mesmo estar cumprindo com sua função de entreter seus usuários enquanto leva a parte pedagógica do desenvolvimento da habilidade da marcha humana.

No link “<https://youtu.be/n8P2AhHwSqU>” tem-se um exemplo de funcionamento curto do jogo, desenvolvido neste projeto, por um voluntário, juntamente com uma explicação rápida sobre o mesmo.

Durante a realização deste projeto, foi possível observar os mecanismos de desenvolvimento de um jogo e como isto pode ser facilmente levado para outras áreas de atuação e, não só, para o entretenimento.

REFERÊNCIAS BIBLIOGRÁFICAS

- APPL, T.; VANNONI, E.; BUSCHMANN, F. **Measuring basal and complex behaviors of rats in automated social home cage systems using IntelliCage for rat technology**. 2008.
- BELDA-LOIS, J.-M.; MENA-DEL HORNO, S.; BERMEJO-BOSCH, I.; et al. Rehabilitation of gait after stroke: a review towards a top-down approach. **Journal of NeuroEngineering and Rehabilitation**, v. 8, n. 1, p. 66, 2011.
- BRASIL, M. D. S. **Diretrizes de Atenção à Reabilitação da Pessoa com com Acidente Vascular Cerebral Diretrizes**. Brasília - DF, 2013.
- CHRISTOU, C.; MICHAEL, D. Aliens versus Humans: Do avatars make a difference in how we play the game? 2014 6th International Conference on Games and Virtual Worlds for Serious Applications, VS-GAMES 2014. **Anais...** . p.1–7, 2014.
- DRACHEN, A.; NACKE, L. E.; YANNAKAKIS, G.; PEDERSEN, A. L. Psychophysiological Correlations with Gameplay Experience Dimensions. , 2010.
- FILKOV, R. Kinect v2 Examples with MS-SDK | RFilkov.com - Technology, Health and More. Disponível em: <<https://rfilkov.com/2014/08/01/kinect-v2-with-ms-sdk/>>. Acesso em: 4/11/2017.
- FLANSBJER, U. B.; HOLMBÄCK, A. M.; DOWNHAM, D.; PATTEN, C.; LEXELL, J. Reliability of gait performance tests in men and women with hemiparesis after stroke. **Journal of Rehabilitation Medicine**, v. 37, n. 2, p. 75–82, 2005.
- FLORES, E.; TOBON, G.; CAVALLARO, E.; et al. Improving patient motivation in game development for motor deficit rehabilitation. Proceedings of the 2008 International Conference in Advances on Computer Entertainment Technology - ACE '08. **Anais...** . p.381, 2008.
- GARCIA MARIN, J.; FELIX NAVARRO, K.; LAWRENCE, E. Serious Games to

Improve the Physical Health of the Elderly: A Categorization Scheme. **International Conference on Advances in Human-oriented and Personalized Mechanisms, Technologies, and Services, CENTERIC 2011**, , n. c, p. 64–71, 2011.

IJSSELSTSIJN, W.; HOOGEN, W. VAN DEN; KLIMMT, C.; et al. Measuring the Experience of Digital Game Enjoyment. In: A. Spink; M. Ballintijn; N. Bogers; et al. (Eds.); *Measuring Behavior 2008 - Proceedings of 6th International Conference on Methods and Techniques in Behavioral Research*. **Anais...** . p.88–89, 2008.

IWABE, C.; DIZ, M. A D. R.; BARUDY, D. P. Analise cinematica da marcha em individuos com Acidente Vascular Encefalico. **Revista Neurociencias**, v. 16, n. 4, p. 292–296, 2008.

KARENY, J.; CARVALHO, S.; FERREIRA, M.; RIBEIRO, M.; VANDENBERGHE, L. A vida após o acidente vascular cerebral na perspectiva dos sobreviventes. **Rev. Eletr. Enf [Internet]**, , n. 4, p. 1–10, 2016.

LANGE, B.; CHIEN-YEN CHANG; SUMA, E.; et al. Development and evaluation of low cost game-based balance rehabilitation tool using the microsoft kinect sensor. 2011 Annual International Conference of the IEEE Engineering in Medicine and Biology Society. **Anais...** . p.1831–1834, 2011.

LEITE, I. C. DE S.; MEIJA, D. P. M. A importância da avaliação minuciosa da marcha em pacientes com distúrbios neurológicos. , p. 1–12, 2013.

LOTARIO, F. A. **Análise do Padrão de Ativação Muscular de Indivíduos Hemiparéticos Pós-AVC em Marcha Assistida por Andador Robótico**. 2015. 2015.

MA, M.; BECHKOUM, K. Serious games for movement therapy after stroke. **2008 IEEE International Conference on Systems, Man and Cybernetics**, p. 1872–1877, 2008.

MACHADO, A. S. **Desenvolvimento de um jogo sério no auxílio da reabilitação funcional do joelho usando o Microsoft Kinect**. 2015. 47-61 f. 2015.

MICROSOFT. Kinect - Desenvolvimento de aplicativos do Windows. Disponível em: <<https://developer.microsoft.com/pt-br/windows/kinect>>. Acesso em: 1/7/2017.

MOUSAVI HONDORI, H.; KHADEMI, M. A Review on Technical and Clinical Impact of Microsoft Kinect on Physical Therapy and Rehabilitation. **Journal of Medical Engineering**, v. 2014, p. 1–16, 2014.

MOYÀ ALCOVER, B.; JAUME-I-CAPÓ, A.; VARONA, J.; MARTINEZ-BUESO, P.; MESEJO CHIONG, A. Use of Serious Games for Motivational Balance Rehabilitation of Cerebral Palsy Patients. **The Proceedings of the 13th International ACM SIGACCESS Conference on Computers and Accessibility**, p. 297–298, 2011.

- NACKE, L. E.; DRACHEN, A.; KUIKKANIEMI, K.; KORT, Y. A. W. DE. Playability and Player Experience Research. **Proceedings of the IEEE**, p. 1–11, 2009.
- OTTOBONI, C.; FONTES, S. V.; FUKUJIMA, M. M. Estudo comparativo entre a Marcha Normal e a de Pacientes Hemiparéticos por Acidente Vascular Encefálico: Aspectos Biomecânicos. **Revista Neurociências**, 2002.
- PASSOS, E. B.; SILVA JR., J. R. DA; RIBEIRO, F. E. C.; MOURÃO, P. T. Tutorial: Desenvolvimento de jogos com unity 3d. **VIII Brazilian Symposium on Games and Digital Entertainment**, p. 1–30, 2009.
- PASSOS, E.; SILVA, J.; CLUA, E. Fast and Safe Prototyping of Game Objects with Dependency Injection. , p. 64–69, 2008.
- SEGURA, D. DE C.; BRUSCHI, F.; GOLIN, T. B.; et al. A evolução da marcha através de uma conduta cinesioterapêutica em pacientes hemiparéticos com sequela de ave. **Arquivos Ciencias da Saúde**, v. 12, n. 1, p. 25–33, 2008.
- SHAH, S.; VANCLAY, F.; COOPER, B. Efficiency, effectiveness, and duration of stroke rehabilitation. **Stroke**, v. 21, n. 2, p. 241–246, 1990.
- SIGN. Management of patients with stroke: Rehabilitation, prevention and management of complications, and discharge planning. **NHS Quality Improvement Scotland**, v. 1, n. 1, p. 108, 2010.
- SILVA, E. D. J. A. DA. Reabilitação após o AVC. , p. 32, 2010.
- SILVA JÚNIOR, S. D. DA; COSTA, F. J. DA. Mensuração e escalas de verificação: uma análise comparativa das escalas de Likert e Phrase Completion. **Seminários de Administração**, 2014.
- Videogame | Michaelis On-line. .Disponível em: <<http://michaelis.uol.com.br/palavra/YkXVd/videogame/>>. Acesso em: 14/9/2017.
- WEISS, P. L.; RAND, D.; KATZ, N.; KIZONY, R. Video capture virtual reality as a flexible and effective rehavilitation tool. **Journal of NeuroEngineering and Rehabilitation**, v. 31, n. 6, p. 12, 2004.
- ZHANG, Z. Microsoft kinect sensor and its effect. **IEEE Multimedia**, Feb. 2012.
- ZYDA, M. From visual simulation to virtual reality to games. **USC Information Sciences Institute**, v. 38, n. 9, p. 25–32, 2005.

APÊNDICES

A – TUTORIAL PARA DESENVOLVIMENTO DO JOGO APRESENTADO

Neste apêndice, apresenta-se um tutorial detalhado para desenvolvimento do jogo apresentado no decorrer deste trabalho. O formato do tutorial será em etapas, unindo informações em textos e também em imagens do próprio software.

A versão utilizada do Unity 3D é a 2017.2.0f3.

1. Inicie o software Unity 3D e crie um novo projeto, como mostra a Figura A.1;

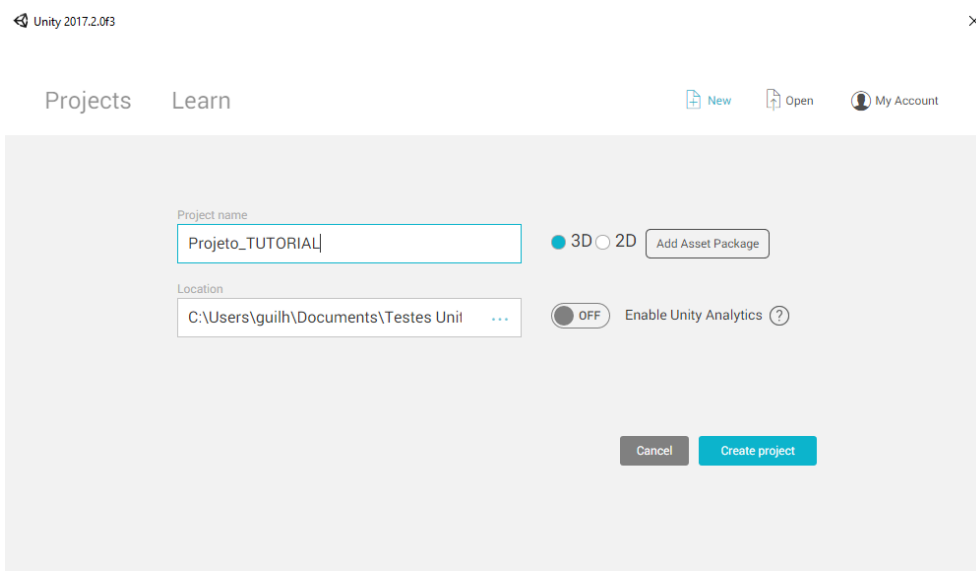


Figura A.1 – Tela inicial de criação de projetos do Unity 3D.

2. Defina um nome para o projeto, selecione “3D”, escolha um local para salvar o projeto e clique em “Create project”;
3. No canto direito superior, clique em “Layout” mude o *layout* para “2 by 3”, conforme mostra a Figura A.2;

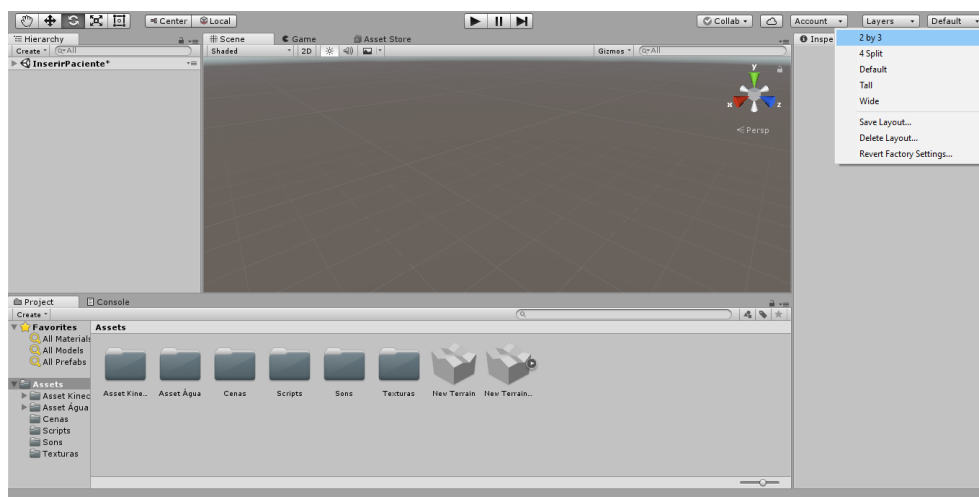


Figura A.2 – Interface principal do Unity 3D.

4. E então, arraste a janela *Project* de forma que a mesma fique abaixo da janela *Hierarchy*. O layout final deve ser como o mostrado na Figura A.3. Se desejar, você pode salvar o layout desenvolvido clicando no mesmo controle para mudança de layout, como no passo 3. Após este passo, a janela deverá ficar como mostra a Figura A.3;

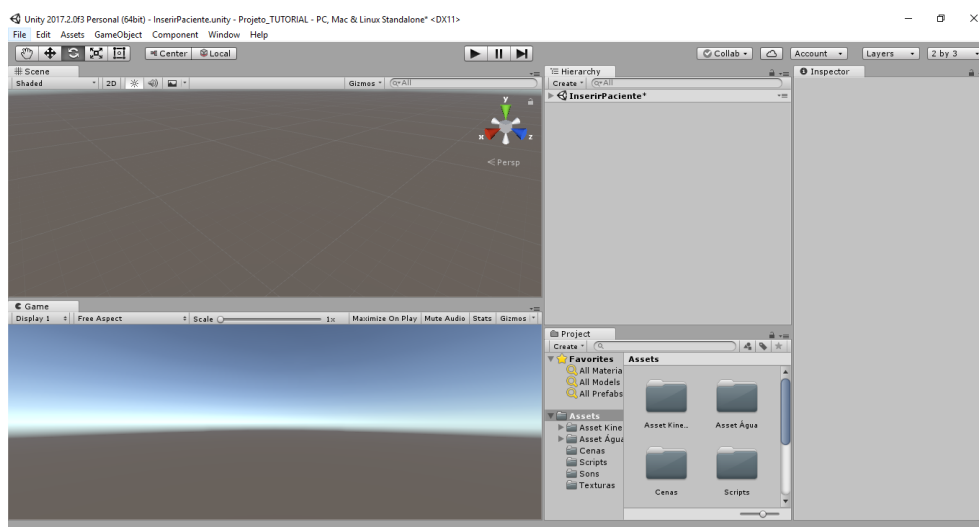


Figura A.3 - Interface modificada do Unity 3D.

5. Clique com o botão direito do mouse na janela *Project* em **Assets** e crie uma nova pasta. Dê o nome a ela de “Asset Kinect”, como mostra a Figura A.4;

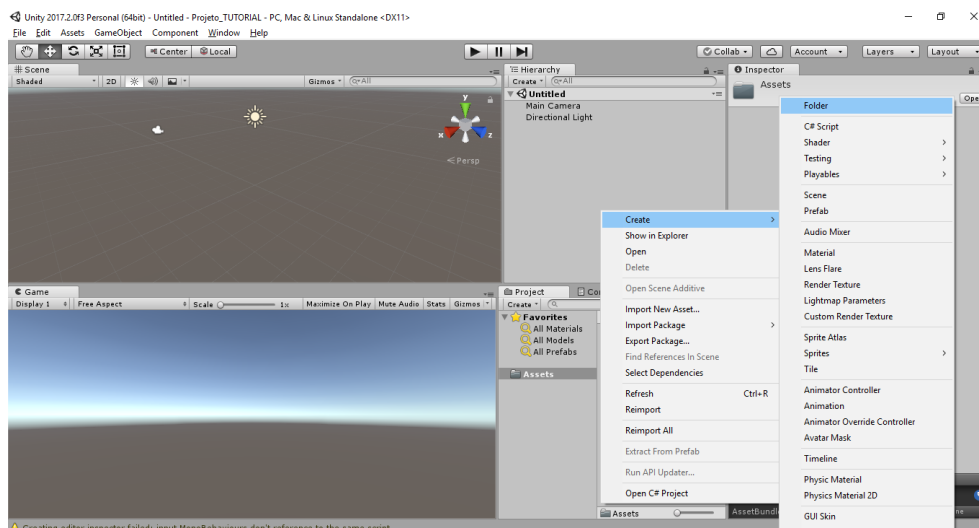


Figura A.4 – Criação de uma pasta dentro do projeto.

6. Faça o tutorial do apêndice B.
7. Selecciona a pasta recém-criada “Asset Kinect” e vá até a barra de ferramentas e clique em *Assets* e então em *Import Package* e então em *Custom Package*, como mostra a Figura A.5;

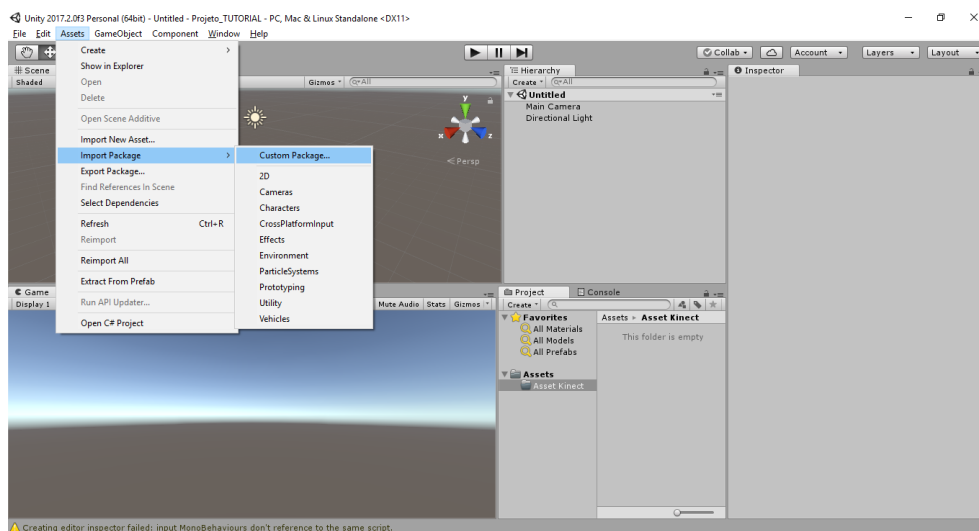


Figura A.5 – Importação de *package* customizados.

8. Encontre o caminho do “Kinect v2 Examples MS-SDK” obtido no passo 4 e clique em “Abrir”;
9. Marque apenas as opções mostradas na Figura A.6 e então clique em “Import”. O *package* em questão possui muitas opções que são utilizadas em diferentes tipos de desenvolvimento de jogos, não sendo necessárias todas para o desenvolvimento deste projeto;

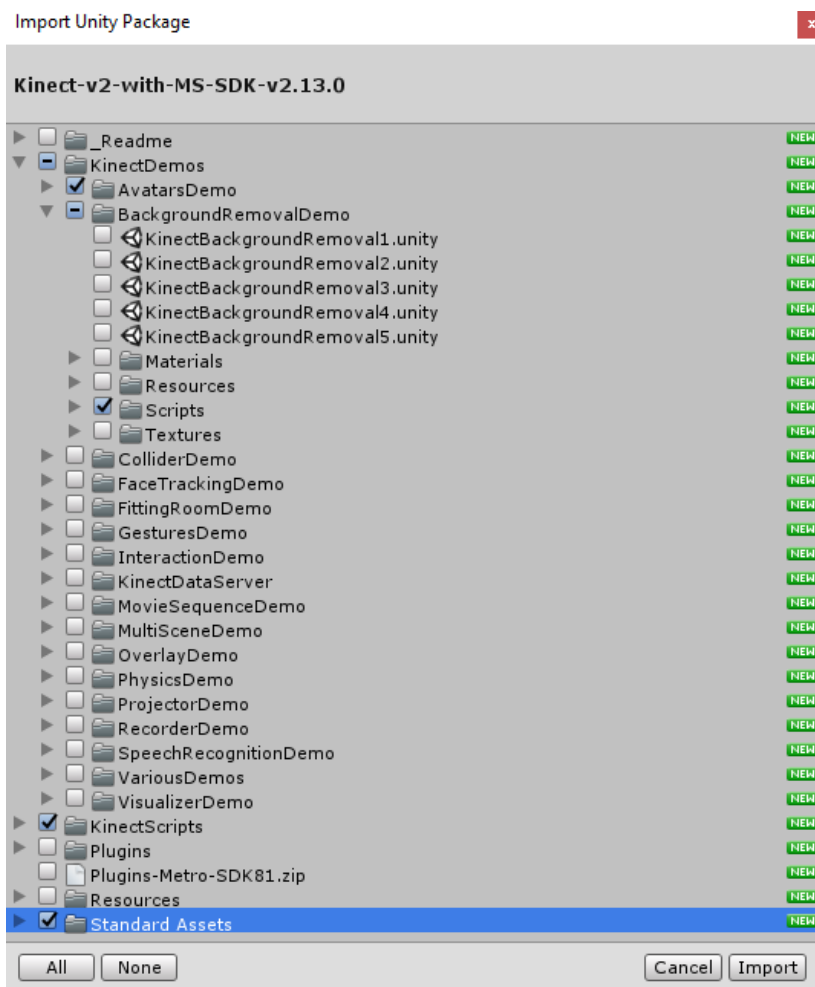


Figura A.6 – Arquivos de importação dentro do *package* “Kinect v2 Examples MS-SDK”.

10. Certifique-se que as novas pastas, geradas na importação do *package*, na janela *Project* estão dentro da pasta criada anteriormente, arrastando-as com o mouse para a pasta “Asset Kinect”;
11. Crie uma nova pasta dentro da pasta “Assets”, como mostrado no passo 5 e dê o nome a ela de “Cenas”;
12. Crie uma nova cena, com o atalho Ctrl+N, e salve-a, com o atalho Ctrl+S, dentro da pasta “Cenas” com o nome de “JogoPrincipal”;
13. Vá até a janela *Hierarchy*, clique em “Create” e então em “3D Object” e então em “Terrain”, como mostrado na Figura A.7;

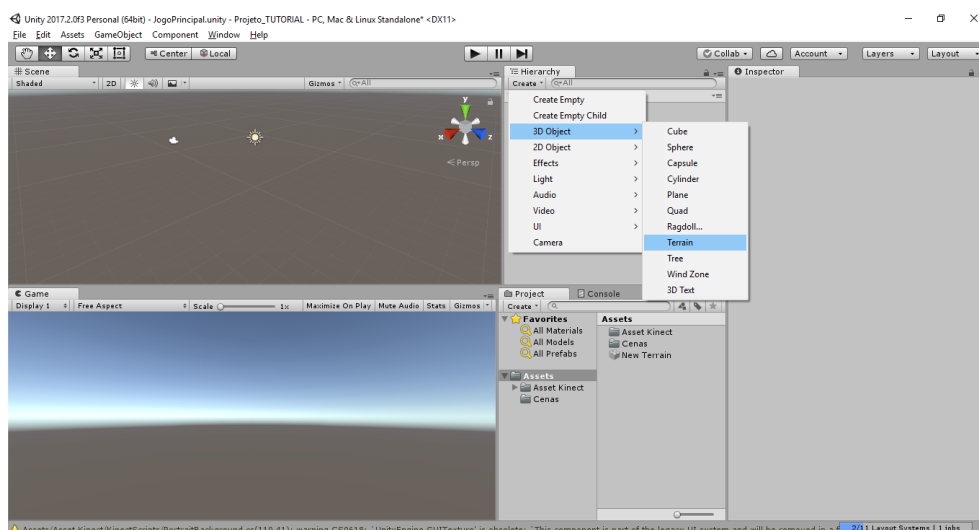


Figura A.7 – Criação de um objeto de jogo tipo “Terrain”.

14. Renomeie o objeto de jogo criado para “Terreno” clicando sobre ele e apertando a tecla F2. Na janela *Hierarchy*, selecione o objeto de jogo em questão e, na janela *Inspector*, vá até o componente *Transform* do objeto e modifique-o para ficar idêntico a Figura A.8. Assim, ficará melhor a visualização do mesmo;

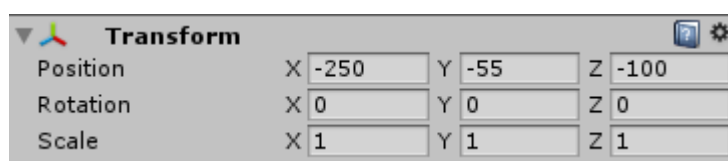


Figura A.8 – Componente *Transform* do objeto de jogo “Terreno”.

15. Para facilitar o trabalho com a manipulação do terreno, recomenda-se posicionar a janela *Scene* para a vista superior clicando no eixo verde, no canto superior da mesma, como mostra a Figura A.9. Pode ser necessário a utilização do comando de zoom para visualização completa do objeto de jogo “Terreno”;

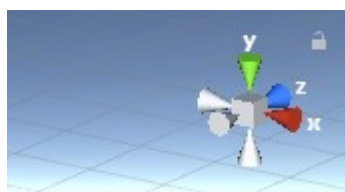


Figura A.9 – Eixo de referência dentro da janela *Scene*.

16. Neste momento, utilize o componente *Terrain* do objeto “Terreno” e faça o relevo desejado para o jogo. Este componente possui ferramentas de “Aumentar e Diminuir” que são o suficiente para produzir o efeito desejado. Selecione a ferramenta e, na janela *Scene*, corra o mouse sobre o plano gerado para criar as deformações ao clicar. Utilize diferentes pincéis de tamanhos variados para gerar padrões interessantes, como na natureza. Oriente-se que o fundo e as laterais sejam similares a montanhas para que o cenário se pareça com um lago, similar a Figura A.10;

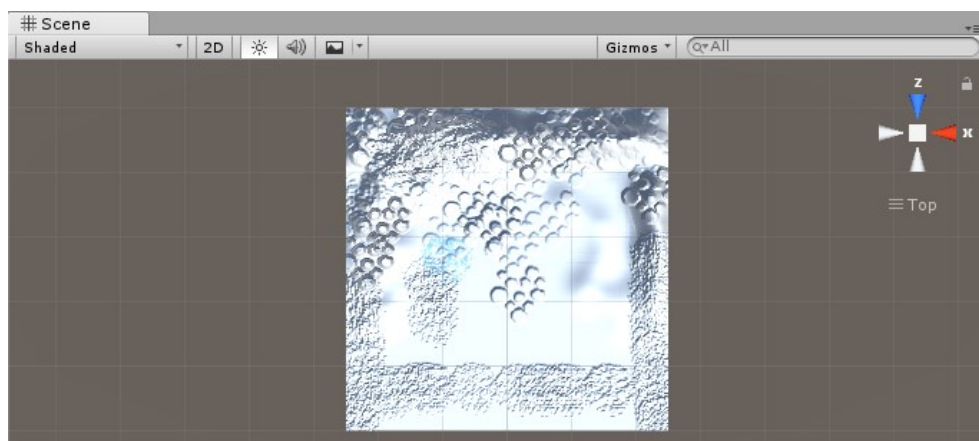


Figura A.10 – Exemplo de terreno construído com as ferramentas da criação de terrenos do Unity 3D.

17. Uma textura nada mais é do que uma imagem que irá ser repetida várias vezes. Desta forma, pode-se escolher a textura que desejar na internet, ao se baixar uma imagem específica. Recomenda-se a utilização de texturas que facilmente se pareçam contínuas, como grama, areia e terra. Para organizarmos melhor, iremos criar uma nova pasta chamada textura e iremos colocar nela as imagens que queremos utilizar como textura. Para inserir imagens nesta pasta, pode-se usar do conceito de clicar e arrastar para a janela *Project* ou pode-se salvar diretamente no diretório onde o jogo está sendo salvo, escolhido no passo 2;
18. Criado os padrões, deve-se, da mesma forma, adicionar as texturas. Com o pincel de texturas selecionado, clique em “Edit Textures...” e então em “Add Texture...”, como mostra a Figura A.11;

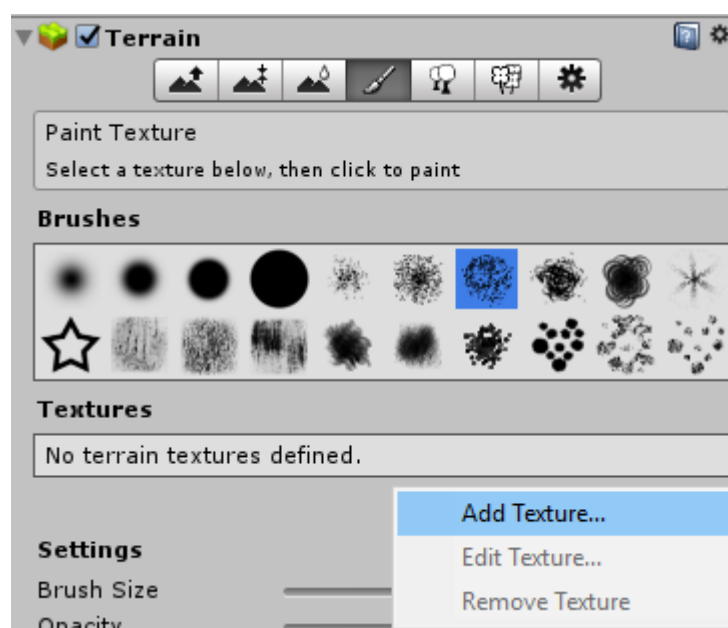


Figura A.11 – Componente “Terrain” e configuração para adicionar texturas.

19. Ao se clicar em “Add Texture...” irá abrir uma caixa de diálogo e, então, deve-se clicar no quadrado mais à esquerda, em “Select”, para que se possa procurar as texturas existentes no projeto, como mostrado na Figura A.12;

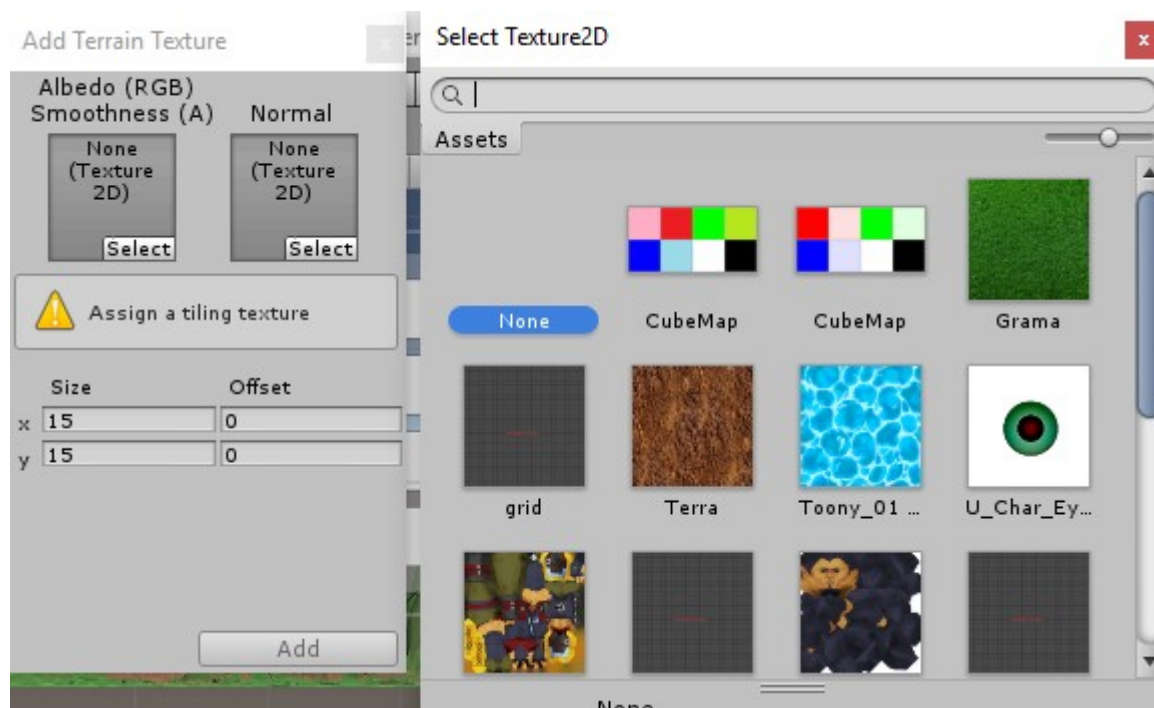


Figura A.12 – Adição de texturas.

20. Após, deve-se clicar duas vezes na textura escolhida e, então, clicar em “Add”;
21. Para aplicar a textura, assim como a deformação para o relevo, deve-se utilizar o mouse na janela *Scene* e selecionar o local no plano onde se quer aplicar a textura. (A primeira textura carregada irá ser aplicada automaticamente sobre todo o terreno. Portanto, recomenda-se carregar primeiramente a textura predominante no terreno. As próximas poderão ser sobrepostas a esta.);
22. Crie uma nova pasta, dentro da pasta “Assets”, e nomeie-a de “Asset Água”. Iremos utilizar esta para importar um novo *asset* para o projeto;
23. Faça o tutorial do apêndice C;
24. Crie um plano e nomeie-o para “Água”, como mostrado na Figura A.13;

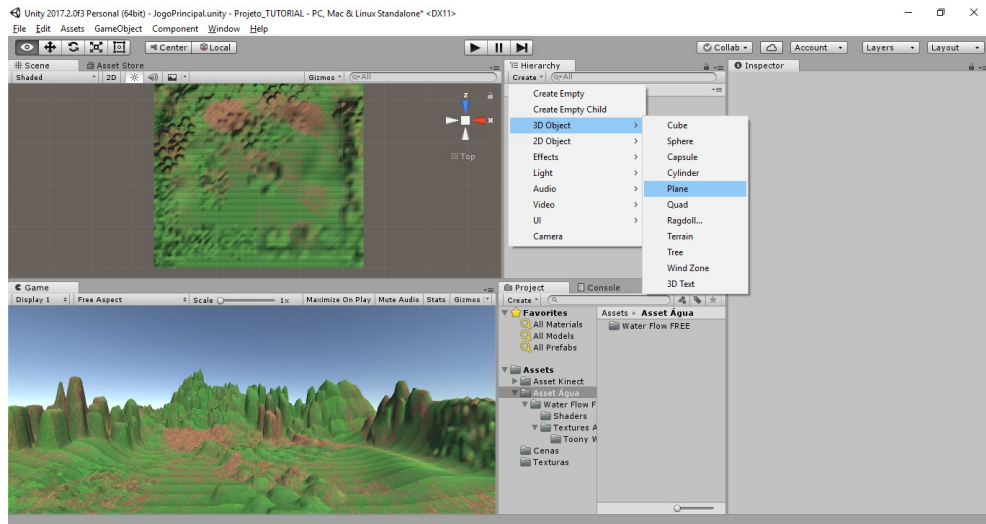


Figura A.13 – Criação de um plano.

25. Modifique o componente *Transform* do mesmo para que fique idêntico ao da Figura A.14. (IMPORTANTE: O valor da coordenada “Y” no componente *Transform* para o vetor “Position” depende da sua formação do terreno. Posicione o mesmo de forma que o cenário fique similar a um lago, como mostra a Figura A.14);

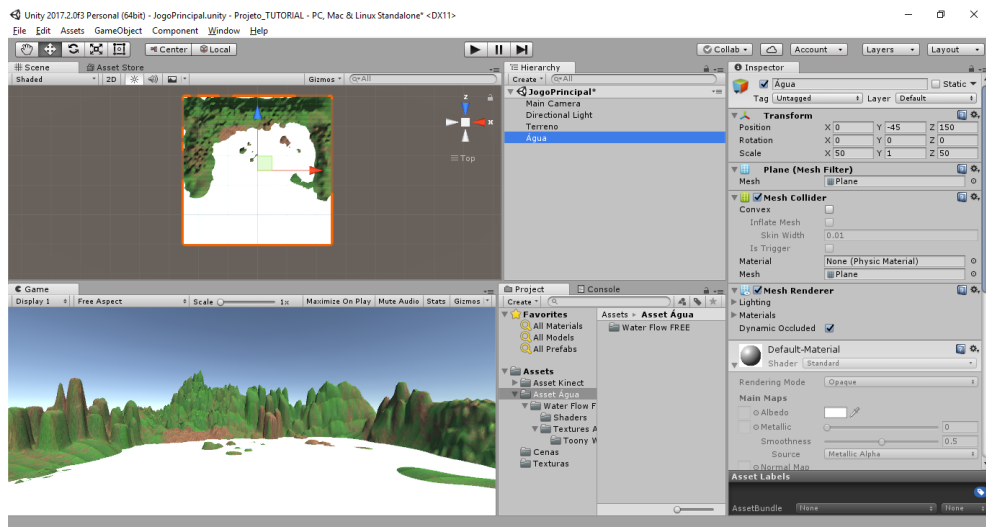


Figura A.14 – Posicionamento do plano da água.

26. Vá até o material importado para a água e procure pelo arquivo “Toony_01_Material (1024x1024px)”, mostrado na Figura A.15.

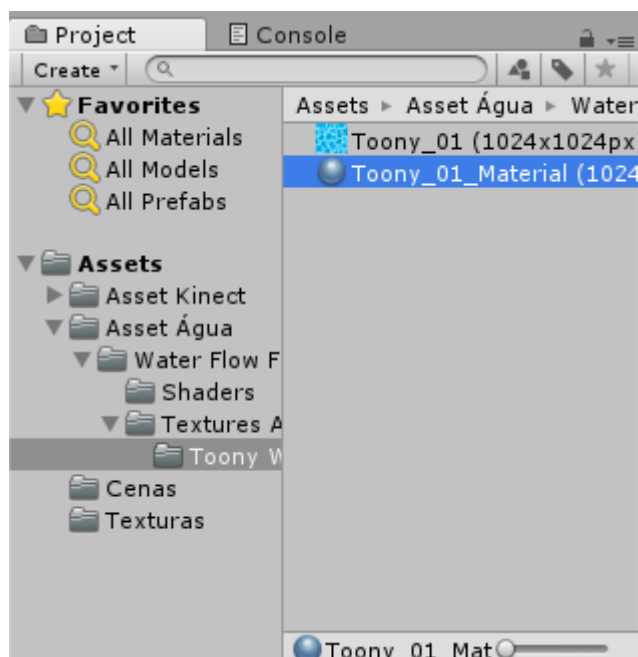


Figura A.15 – Material para textura de água.

27. Encontrado, arraste-o até o objeto de jogo “Água” recém-criado e solte, com o mouse, como na Figura A.16.

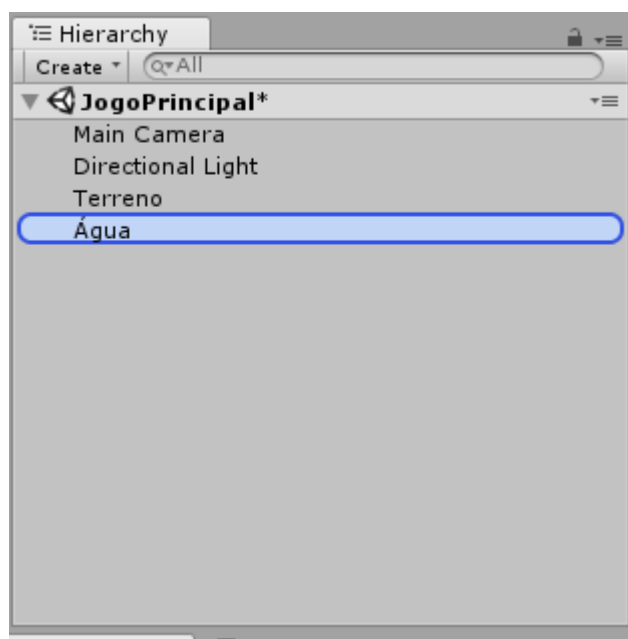


Figura A.16 – Exemplo da janela *Hierarchy* para quando soltar o material sobre o objeto “Água”.

28. Crie uma nova pasta e dê o nome de “Scripts” a ela.
29. Dentro da pasta “Scripts”, clique com o botão direito na janela *Project* e crie um *script* em C#, como mostra a Figura A.17, e dê o nome do *script* de “Mare”.

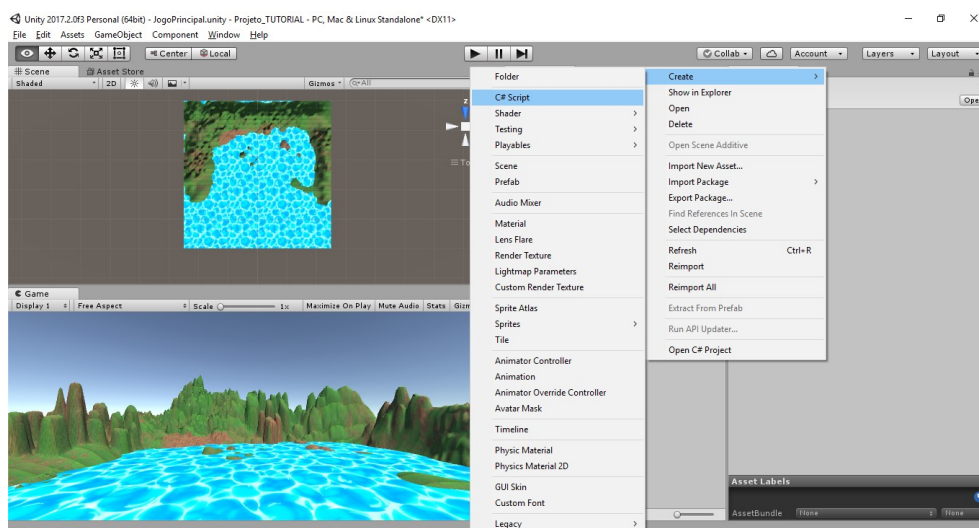


Figura A.17 – Criação de um *script* em C#.

30. De forma a padronizar o desenvolvimento e o acompanhamento deste tutorial, selecione o menu preferências indo em “Edit” e então “Preferences...”, como mostrado na Figura A.18;

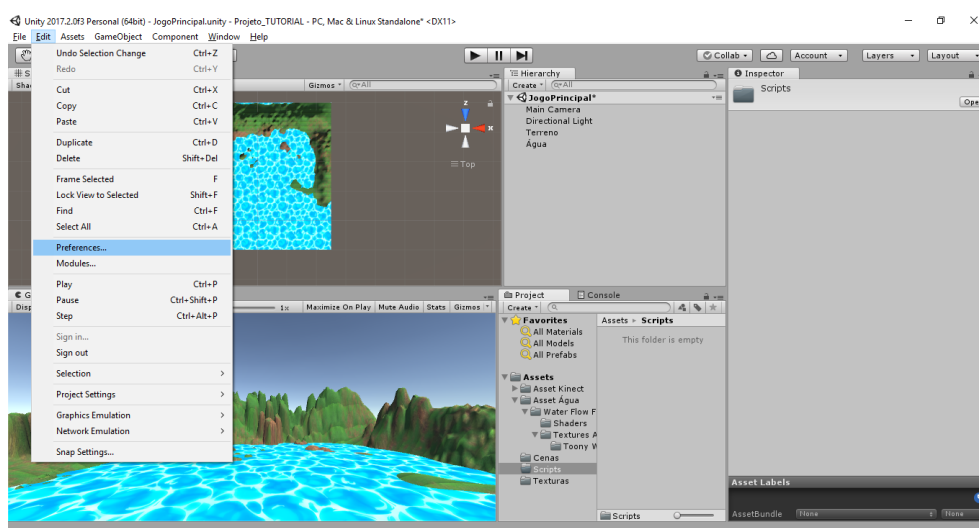


Figura A.18 – Acesso ao menu “Preferences...”.

31. Vá até a opção “External Tools” e, no campo “External Script Editor”, selecione “MonoDevelop (built-in)”, como mostrado na Figura A.19. Isto fará com que o editor de *scripts* será o software MonoDevelop, um software distribuído juntamente com a instalação do Unity 3D. Desta forma, garante-se a presença do mesmo na máquina do desenvolvedor;

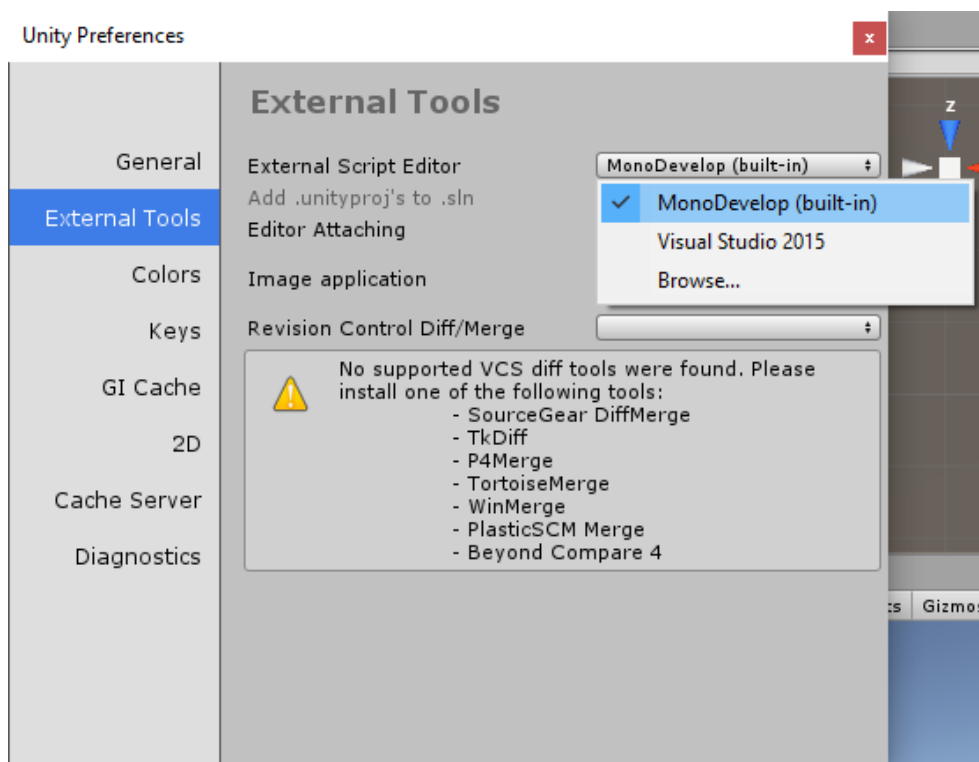


Figura A.19 – Seleção do software “MonoDevelop” para edição dos *scripts*.

32. Clique duas vezes no *script* recém-criado “Mare”. Ele irá ser aberto no software MonoDevelop, como mostra a Figura A.20;

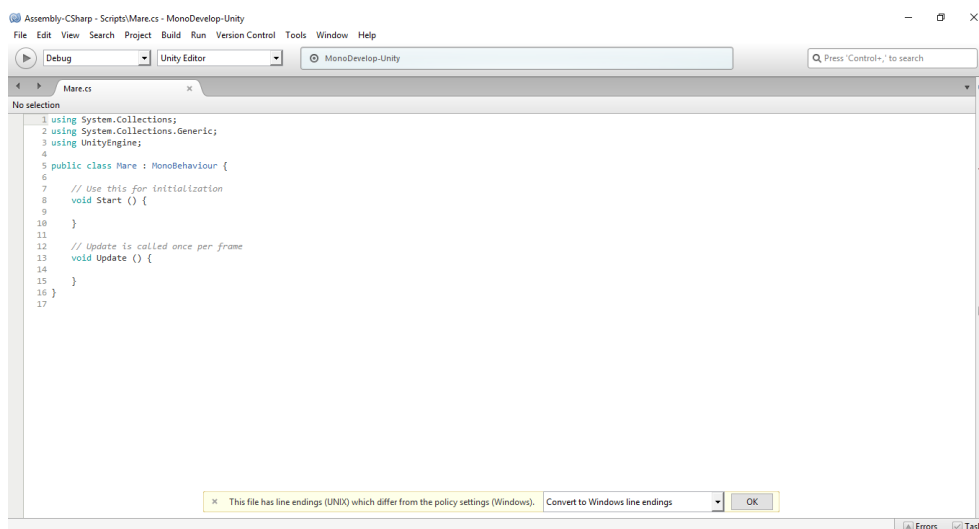


Figura A.20 – MonoDevelop.

33. Apague todo o conteúdo mostrado acima e copie o conteúdo do *script* “Mare.cs”, presente no apêndice I, para o mesmo. Salve o *script* e volte para a tela do Unity 3D e aguarde alguns segundos até o sinal mostrado na Figura A.21, que está no canto inferior direito, desaparecer (não é necessário fechar o MonoDevelop);



Figura A.21 – Sinal de compilação do *script*.

34. Clique no objeto de jogo “Água”, vá até a janela *Inspector* e clique em “Add Component”, como mostrado na Figura A.22;

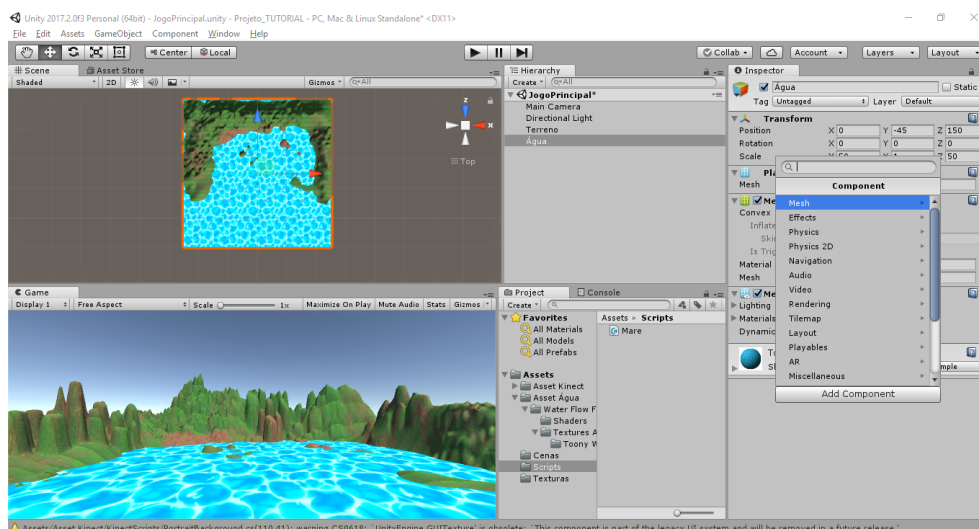


Figura A.22 – Adicionar um novo componente a um objeto de jogo.

35. No campo de busca, escreva “Mare” e, quando encontrar o *script* recém-criado, adicione-o com o botão “Enter” ou com o clique do mouse, como mostra a Figura A.23;

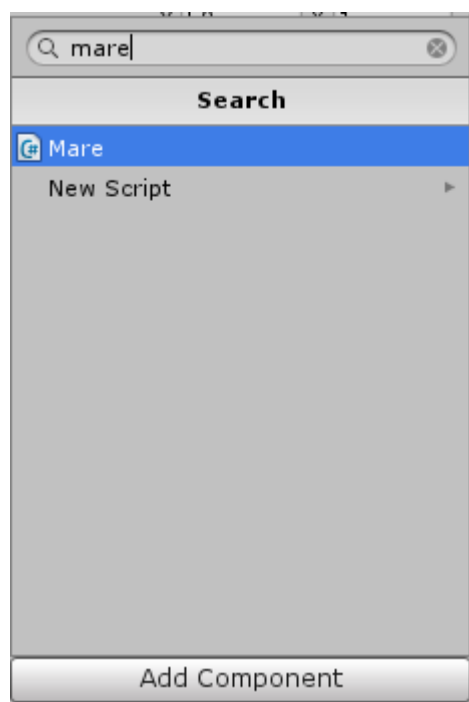


Figura A.23 – Busca de um componente pelo nome do mesmo.

36. A iluminação fica a critério de desenvolvimento. Para criar novos pontos de luz, ou outros tipos de iluminação, siga a Figura A.24.

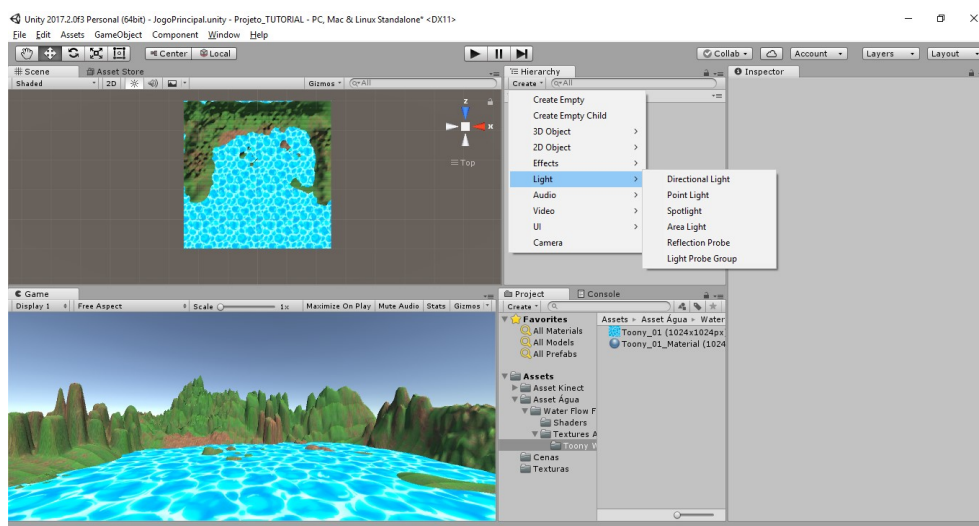


Figura A.24 – Criação de componentes de luminosidade.

37. E, as configurações, podem ser feitas manualmente na janela *Inspector* ao clicar no objeto que se deseja configurar.
38. Na janela *Hierarchy*, clique em “Create” e crie um objeto de jogo vazio, como mostrado na Figura A.25 e dê nome a ele de “Musica”;

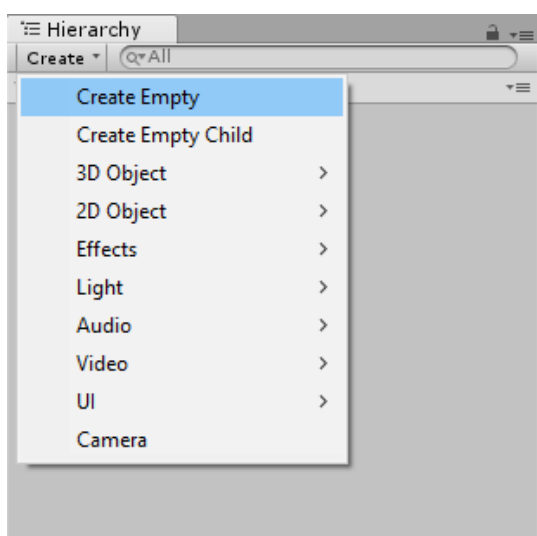


Figura A.25 – Criação de um objeto de jogo vazio.

39. Clique no objeto recém-criado e, na janela *Inspector*, clique em “Add Component” e, usando o campo de busca, procure por “Audio Source” e, então, adicione-o com o botão “Enter” ou com o clique do mouse;
40. Crie uma pasta dentro da pasta “Assets” e dê o nome a ela de “Sons”;
41. Assim como para as texturas, os sons são escolhas do desenvolvedor. São facilmente encontrados na internet, podendo ser sons ambientes ou, até mesmo, músicas. (Um exemplo de utilização possível é o áudio do link <https://www.dl-sounds.com/royalty-free/the-coast/> - acesso em 27/01/2018 com um som ambiente para a movimentação da água). O componente “Audio Source”, mostrado na Figura A.26, dá a possibilidade de

configurar para que o áudio inserido toque repetidamente (opção “Loop”) e que o mesmo comece assim que a cena se iniciar (opção “Play On Awake”). Para inserir arquivos de áudio, basta realizar os mesmos procedimentos realizados para texturas, baixando-os e salvando-os na pasta destinada ao mesmo dentro do projeto do jogo e selecionando-os ao se clicar no botão a direita da configuração “AudioClip”. Os arquivos de áudio existentes no projeto irão aparecer em uma caixa de diálogo para escolha do desenvolvedor;

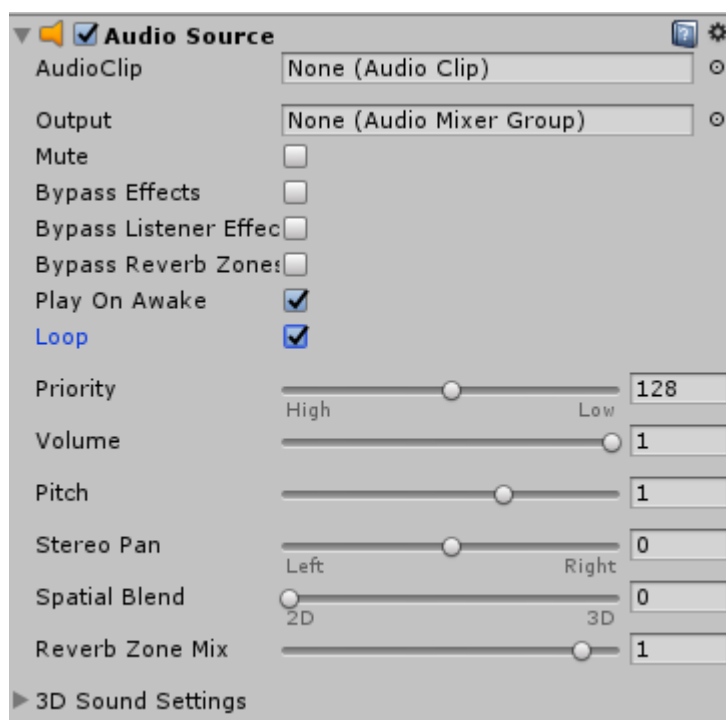


Figura A.26 – Componente “Audio Source” para manipulação de áudios.

42. Neste momento; tem-se o cenário completo para todas as 3 cenas. Portanto, vá até a pasta “Cenas” e duplique (Ctrl+D) a cena em questão, “JogoPrincipal” dentro da pasta “Cenas”, para que fiquem 3 delas. Nomeie uma das novas cenas como “MenuPrincipal” e a outra como “InserirPaciente”;
43. Abra a cena “MenuPrincipal” clicando duas vezes sobre ela;
44. Deve-se agora organizar as cenas do jogo. Para isto, vá “File” e então em “Build Settings”, como mostra a Figura A.27;

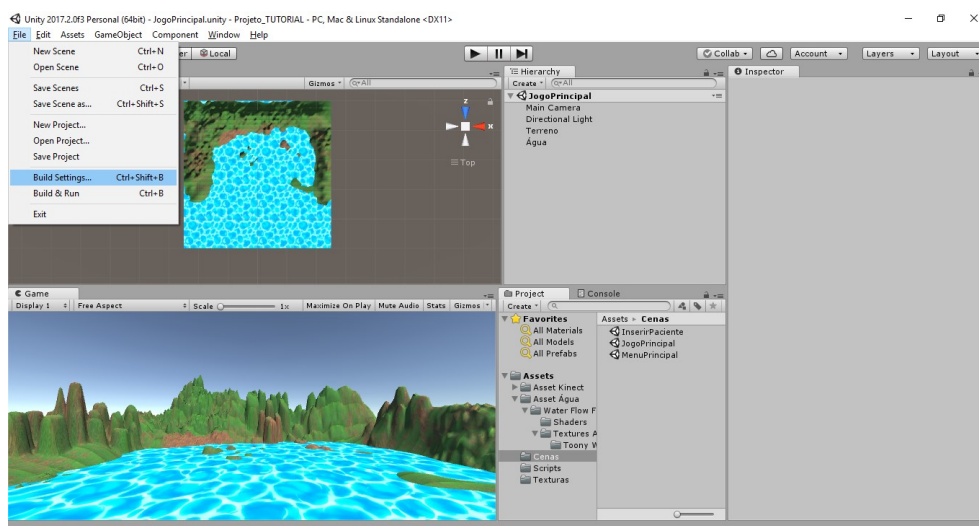


Figura A.27 – Acessando as configurações de construção de jogo.

45. Clique em “Add Open Scenes”. Isto adicionará a cena atual nas cenas de construção do jogo e dará a ela um valor inteiro de identificação. Ao executar esta adição, o valor dado é o menor valor depois do último. Portanto, deve-se executar esta adição, do passo 43 ao 45, na ordem “MenuPrincipal”, “InserirPaciente” e então “JogoPrincipal”. Desta forma, os valores de identificação serão 0, 1 e 2. Padronizados nos *scripts* desenvolvidos. E então, feche a janela de configurações de construção de jogo;
46. Agora, vamos completar a cena “MenuPrincipal”. Na pasta “Cenas”, abra esta cena clicando duas vezes nela;
47. Na janela *Hierarchy*, clique em “Create” e crie um *Canvas*, como mostrado na Figura A.28;

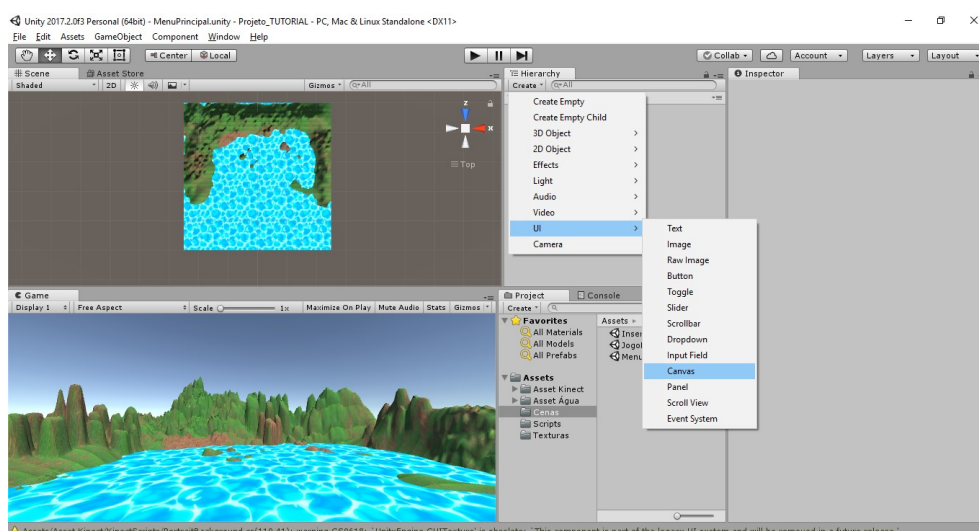


Figura A.28 – Criação de um painel *Canvas*.

48. Na janela *Scene*, mude a visualização da cena para 2D no botão realçado na Figura A.29. Isto facilitará o trabalho nesta cena;

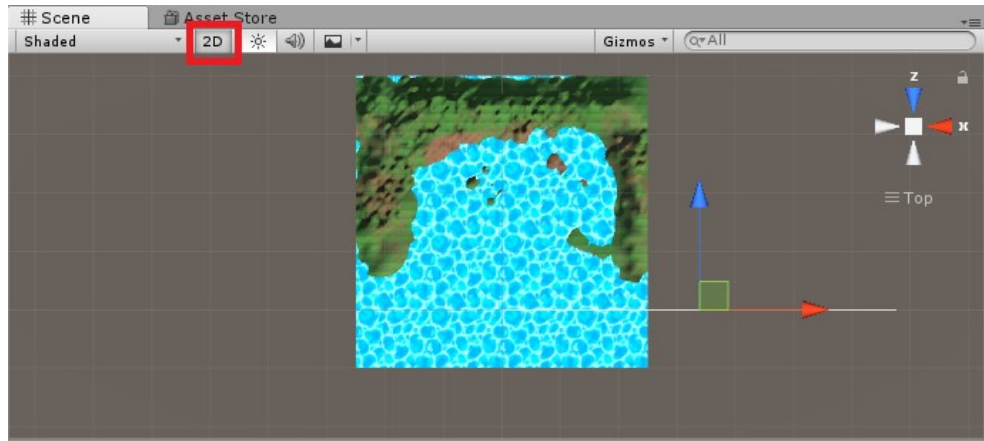


Figura A.29 – Modificação da visão da cena para 2D.

49. Clique com o botão direito sobre o objeto recém-criado *Canvas* e então clique em “UI” e então em “Panel”, como mostrado na Figura A.30 para criar um painel como sendo filho do objeto *Canvas*. Nomeie-o como “menuTitulo”;

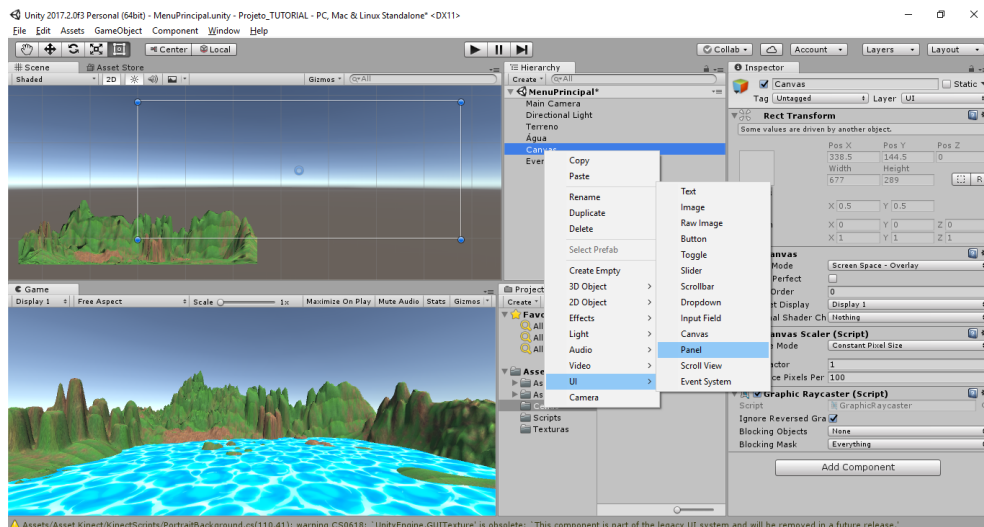


Figura A.30 – Criação de um painel dentro do painel *Canvas*.

50. No componente “Rect Transform” do painel “menuTitulo”, configure-o para que fique idêntico a Figura A.31. Isto faz com que existam bordas para o painel criado, dando uma margem de respiro para a interface;



Figura A.31 – Componente *Rect Transform* do objeto de jogo “menuTitulo”.

51. Clique no objeto “menuTitulo” e, na janela *Inspector* clique em “Add Component”. Procure pelo componente “Vertical Layout Group” e adicione-o ao objeto;
52. No objeto “menuTitulo”, configure o componente “Vertical Layout Group” para que fique idêntico ao da Figura A.32. Este componente organiza os objetos presentes no painel. Ele centraliza os mesmos tanto verticalmente quanto horizontalmente com os valores de margens e de espaçamentos escolhidos;

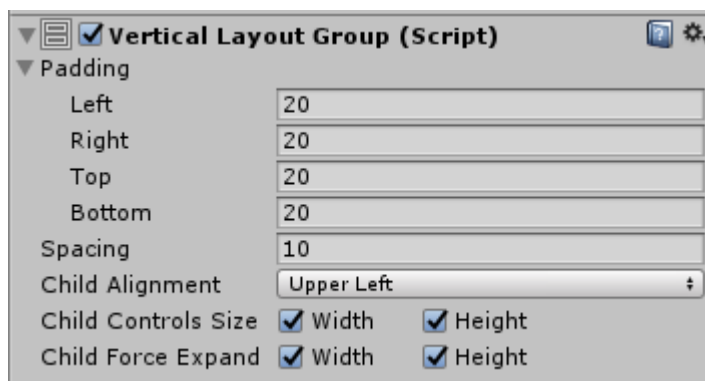


Figura A.32 – Componente *Vertical Layout Group* do objeto de jogo “menuTitulo”.

53. Duplique (Ctrl+D) o objeto “menuTitulo” de forma que existam 3 painéis e, então, nomeie os dois novos, na respectiva ordem: menuAjuda, carregando..., como mostrado na Figura A.33;

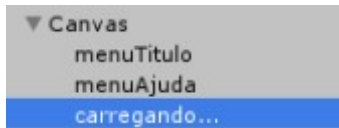


Figura A.33 – Representação dos painéis dentro do painel *Canvas*.

54. Agora, para trabalhar cada um dos painéis separadamente, deve-se esconder os outros. Para isto, clique sobre o painel a ser escondido e, na janela *Inspector*, clique na caixa de seleção destacada na Figura A.34. Deixe ativado apenas o objeto “menuTitulo” dentre os painéis criados;

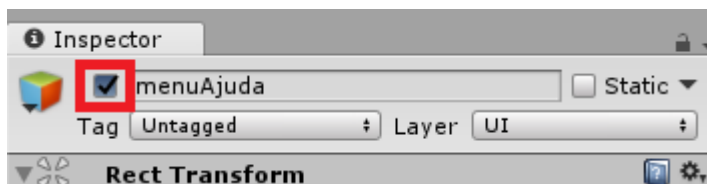


Figura A.34 – Ativação ou desativação de objetos de jogo.

55. Clique com o botão direito no objeto “menuTitulo” e, então, clique em “UI” e clique, finalmente, em “Button”, como mostra a Figura A.35;

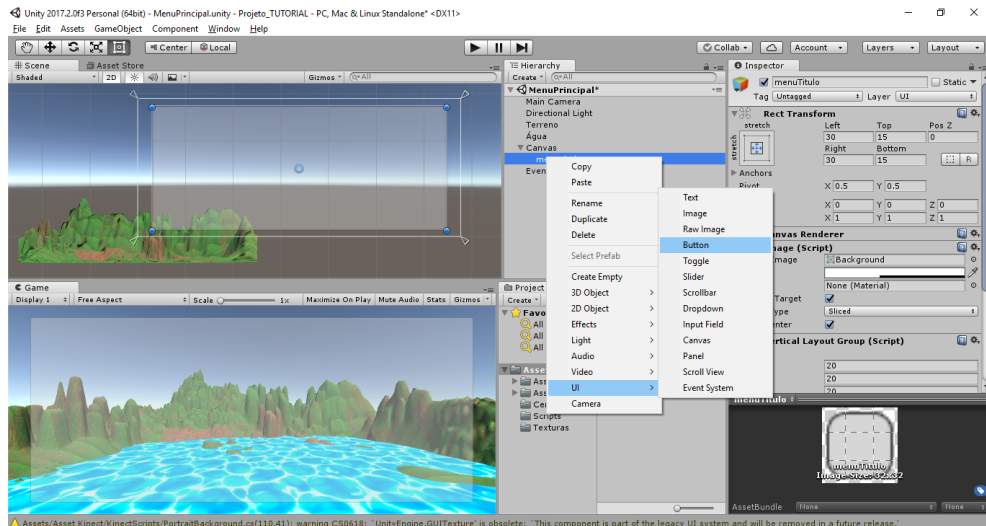


Figura A.35 – Criação de um botão.

56. Duplique este botão criado até que se tenha quatro botões na tela;
57. Renomeie os botões criados para os seguintes nomes, exatamente na ordem apresentada aqui: iniciar, ajuda, sobre e sair. O resultado final deve ser idêntico a Figura A.36;

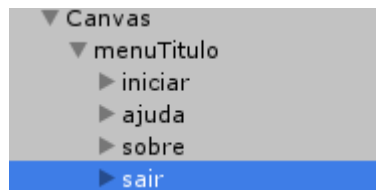


Figura A.36 – Botões do objeto de jogo “menuTitulo”.

58. Selecione o botão iniciar, clique em seu objeto filho “Text” e, no componente “Text (Script)”, edite o texto padrão “Button” para o nome do botão em questão, como mostra a Figura A.37. Marque também a opção “Best Fit”, no mesmo componente e defina os valores de mínimo para 10 e de máximo para 30. Isto faz com que as palavras no painel se adaptem a diferentes formatos e tamanhos de tela;

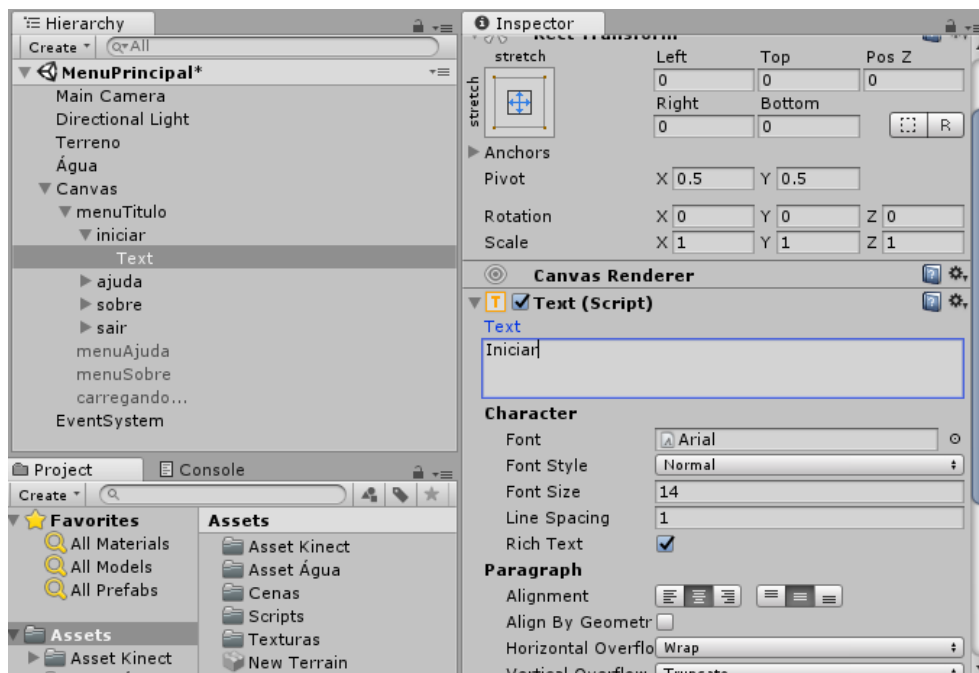


Figura A.37 – Componente “Text (Script)” do objeto de jogo “Text”.

59. Repita o passo 58 para os outros botões;
60. Faça o passo 54 para o objeto “menuTitulo” e repita o mesmo para o “menuAjuda”. Neste caso, como o “menuAjuda” está desativado, ele será ativado;
61. Clique com o botão direito sobre o “menuAjuda” e, crie um texto como objeto filho, como mostra a Figura A.38;

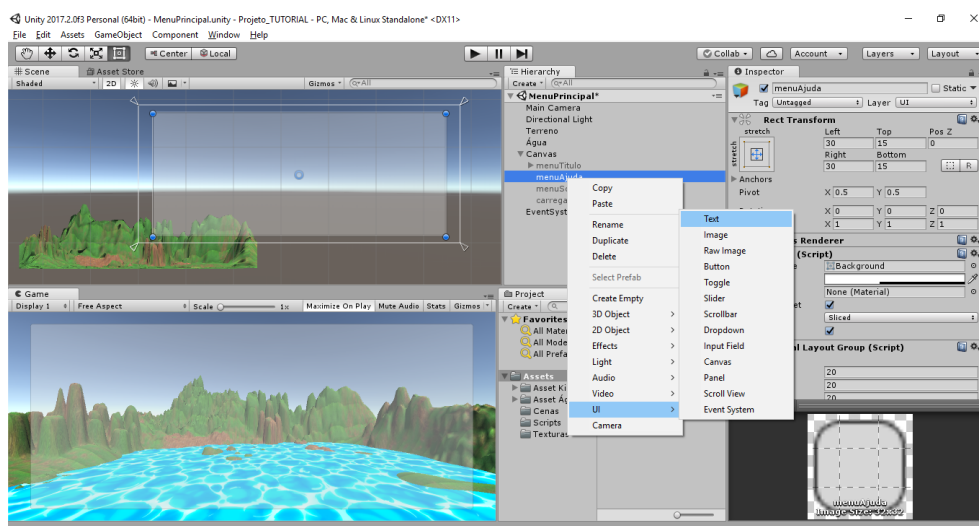


Figura A.38 – Criação de um objeto de jogo do tipo texto dentro de outro objeto de jogo do tipo botão.

62. Repita o passo 61 (após este passo, ter-se-á criado dois objetos “Text”);
63. Clique com o botão direito sobre o “menuAjuda”, e crie um botão, como explicado no passo 55;

64. Caso a configuração de organização não tenha acontecido automaticamente, no objeto “menuAjuda”, modifique qualquer parâmetro do componente “Vertical Layout Group” e volte para o valor anterior;
65. Renomeie o primeiro “Text” para “textoTitulo” e em seu componente “Text (Script)”, nomeie-o para “Ajuda”. Em “Paragraph”, na configuração “Alignment”, escolha centralizar horizontalmente e verticalmente. Selecione a opção “Best Fit” e, defina o valor mínimo para 10 e o valor máximo para 40;
66. Renomeie o “Text (1)” para “textoAjuda” e, em seu componente “Text (Script)”, insira o texto que deseja incluir para auxílio do usuário. Em “Paragraph”, na configuração “Alignment”, escolha centralizar horizontalmente e verticalmente;
67. Renomeie o botão para “voltarAjuda” e, em seu objeto filho “Text”, no componente “Text (Script)”, escreva “Voltar”;
68. Duplicue o objeto “menuAjuda” e nomeie o novo objeto como “menuSobre”;
69. A partir do passo 54, repita o procedimento de forma que o objeto “menuSobre” seja o objeto ativado;
70. Renomeie o objeto “textoAjuda” dentro do objeto “menuSobre” para “textoSobre” e o objeto “voltarAjuda” para “voltarSobre”;
71. No objeto “textoTitulo” dentro do objeto “menuSobre”, vá até o componente “Text (Script)” e substitua o texto “Ajuda” para “Sobre”;
72. No objeto “textoSobre” dentro do objeto “menuSobre”, vá até o componente “Text (Script)” e substitua o texto específico para auxílio para um texto específico para informações sobre o desenvolvimento do jogo;
73. Repita o passo 54 para que o objeto “carregando...” fique ativo, enquanto os outros painéis fiquem inativos;
74. Repita o passo 61 para criar um objeto de texto dentro do painel “carregando...” e nomeie-o para “textoCarregando”;
75. No componente “Text (Script)” do objeto “textoCarregando”, modifique o texto para “Carregando...” e em “Paragraph”, na configuração “Alignment”, escolha centralizar horizontalmente e verticalmente. Selecione a opção “Best Fit” e, defina o valor mínimo para 10 e o valor máximo para 40;
76. Crie um objeto do tipo *slider* como filho do objeto de jogo “carregando...”, conforme mostra a Figura A.39, e nomeie-o para “Barra”;

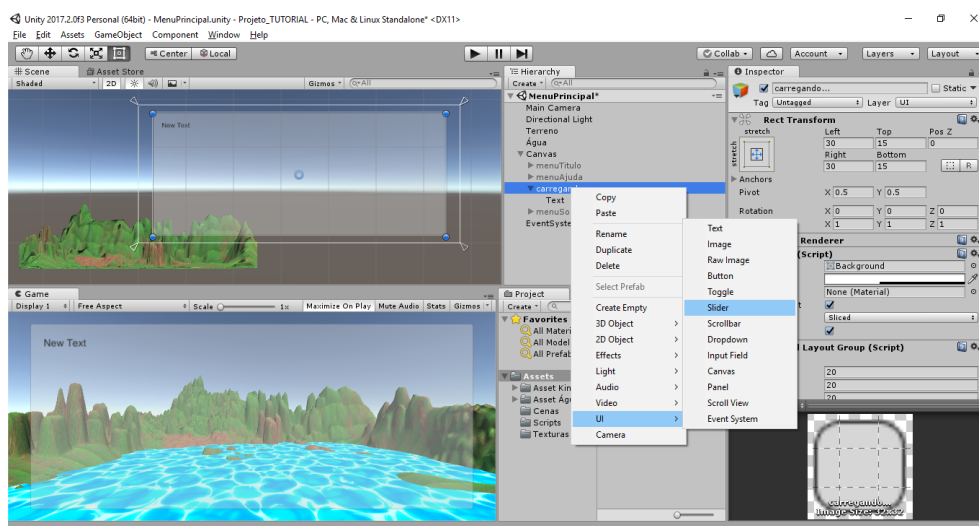


Figura A.39 – Criação de um objeto de jogo do tipo “Slider”.

77. Caso seja necessário, execute o passo 64 no objeto “carregando...”;
78. No objeto “Barra”, expanda-o e, então, exclua o objeto filho “Handle Slide Area”, uma vez que o mesmo não é necessário nesta abordagem;
79. Modifique o componente “Rect Transform” do objeto “Fill”, filho do objeto de jogo “Fill Area” para que fique idêntico ao da Figura A.40;

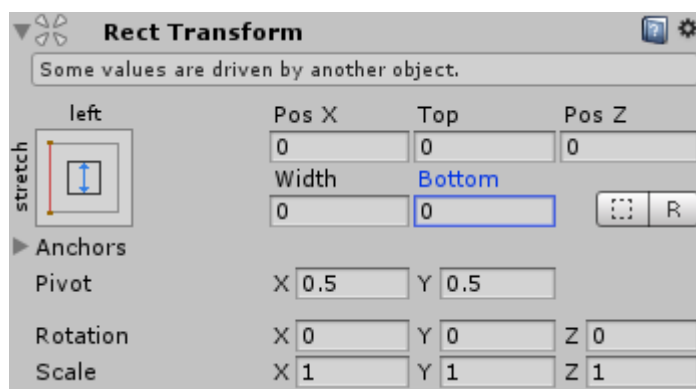


Figura A.40 – Componente *Rect Transform* do objeto de jogo “Fill”.

80. Clique com o botão direito no objeto “Barra”, e então clique em “UI” e crie um objeto de texto ao clicar em “Text”;
81. Clique no objeto criado “Text”, em seu componente “Text (Script)” Em “Paragraph”, na configuração “Alignment”, escolha centralizar horizontalmente e verticalmente. Marque a opção “Best Fit” e defina o valor mínimo para 10 e o valor máximo para 40;
82. Faça com que o objeto “menuTitulo” fique ativo, conforme o passo 54;
83. Repita o passo 38 para criar um objeto de jogo vazio e nomeie o mesmo para “LevelLoader”;
84. Repita o passo 29 para criar um *script* C# (dentro da pasta “Scripts”) e nomeie-o como “LevelLoader”. Feito isto, abra o *script* recém-criado, apague o conteúdo padrão e copie o código do apêndice H para o *script* e salve. Volte para a tela do Unity 3D;

85. Repita o passo 34 e 35 para adicionar o *script* “LevelLoader” como um componente do objeto de jogo “LevelLoader”;
86. Ao adicionar o *script* ao objeto “LevelLoader”, temos uma visão da janela *Inspector* como mostrado na Figura A.41;

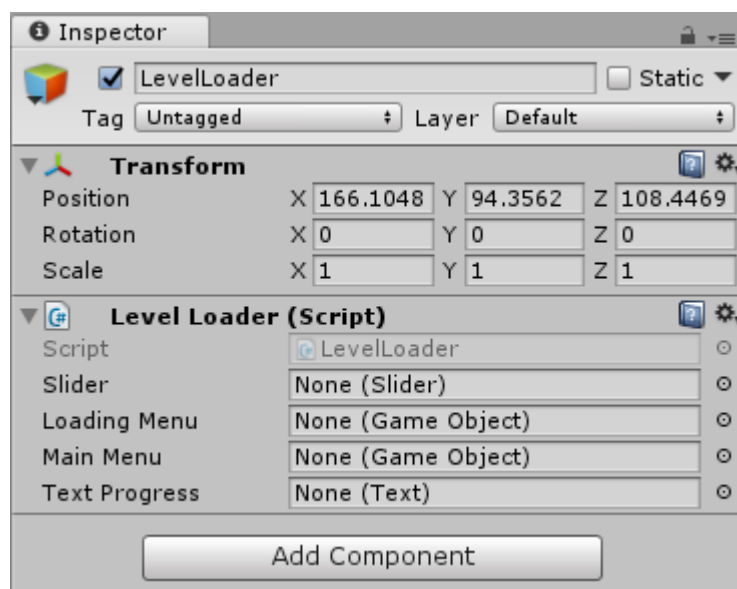


Figura A.41 – Inserção do novo componente “Level Loader (Script)” em um objeto de jogo.

87. Vá até a janela *Hierarchy*, dentro do objeto de jogo “Canvas”, dentro o objeto filho “carregando...”, e encontre o objeto “Barra”, como mostra a Figura A.42;

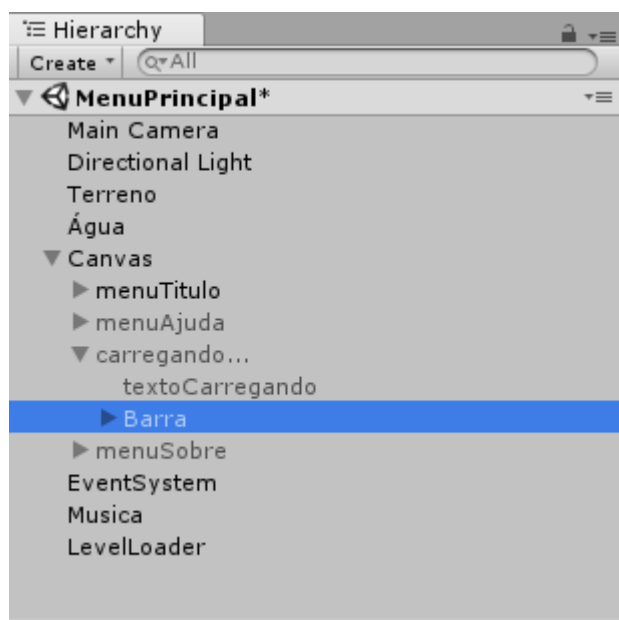


Figura A.42 – Local de acesso ao objeto “Barra”.

88. Sem clicar no objeto “Barra”, arraste-o com o mouse para o campo à direita do nome “Slider” dentro do componente “Level Loader (Script)”, destacado na Figura A.43;

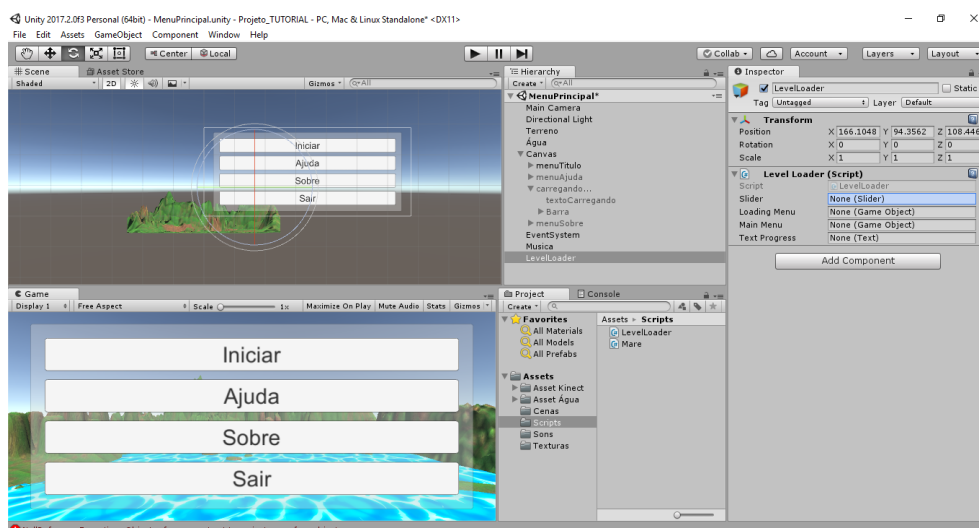


Figura A.43 – Inserção de objetos de jogo para serem utilizados pelas variáveis dentro de um *script*.

89. Repita o procedimento no passo 87 e 88 para completar as variáveis do componente “Level Loader (Script)”. Mas desta vez, a variável “Loading Menu” deverá ser completada com o objeto “carregando...”, a variável “Main Menu”, com o objeto de jogo “menuTitulo” e a variável “Text Progress” com o objeto “Text” filho do objeto “Barra”. Por fim, temos uma configuração idêntica à da Figura A.44;

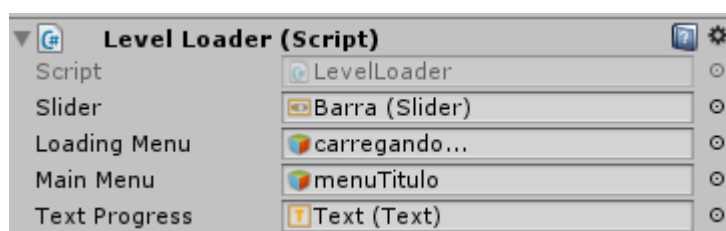


Figura A.44 – Configuração completa do *script* “LevelLoader.cs”.

90. Agora vamos fazer o controle dos botões. Repita o passo 29 e crie um *script* e nomeie-o como “Sair”. Feito isto, abra o *script* recém-criado, apague o conteúdo padrão e copie o código do apêndice L para o *script* e salve. Volte para a tela do Unity 3D;
91. Selecione o botão “Sair” na janela *Hierarchy*;
92. Repita o passo 34 e 35 para fazer com que o *script* “Sair” recém-criado seja componente do objeto de jogo “Sair”;
93. Na janela *Inspector*, no componente “Button (Script)”, vá até a configuração “On Click ()” e, então, aperte no botão em forma de “+”, destacado na Figura A.45;

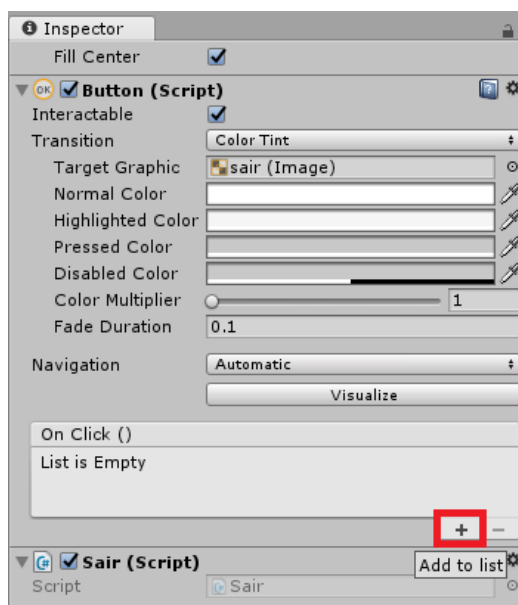


Figura A.45 – Destaque para a inserção de eventos para os objetos de jogo do tipo botão.

94. Ao executar o passo anterior, o Unity 3D cria um evento para que o mesmo aconteça ao clicar do botão. Vamos agora configurar este evento. Para isto, basta arrastar o componente “Sair (Script)” presente abaixo da configuração “On Click ()”, como mostra a Figura A.45 para o campo presente no evento criado;
95. Clique em “No Function”, vá até “Sair” e selecione a opção “sairJogo ()”, como mostra a Figura A.46. Como a função “sairJogo ()” não possui parâmetros de entrada. Ela já será ativada quando o botão “Sair” for pressionado;

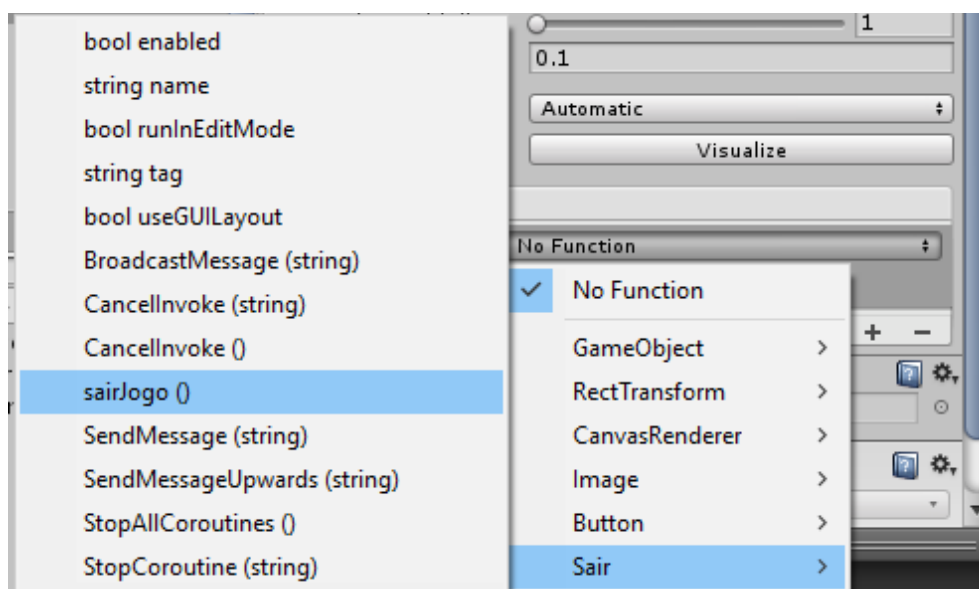


Figura A.46 – Seleção da função “sairJogo ()” para o evento ao clicar.

96. Selecione o botão “Voltar” do objeto “menuAjuda” (lembre-se que o mesmo está nomeado como “voltarAjuda”). E então execute o passo 93 duas vezes. Após isto, teremos a configuração “On Click ()”, como mostra a Figura A.47;

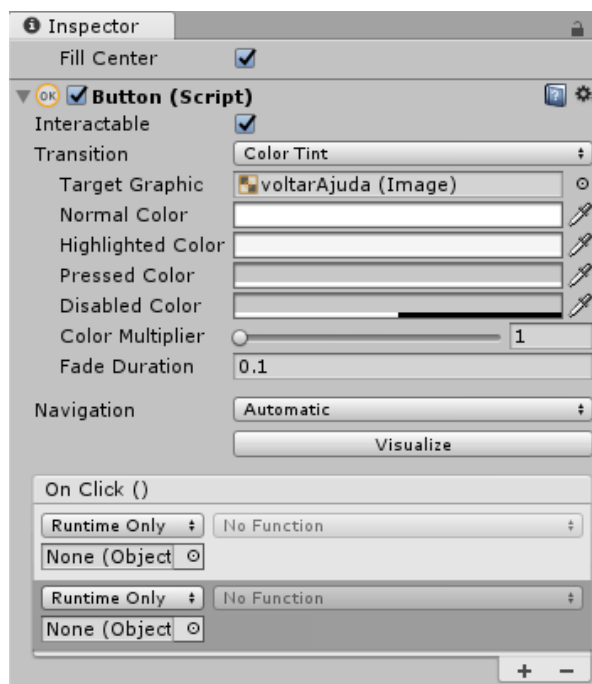


Figura A.47 – Criação de dois eventos para ocorrerem durante um mesmo clique.

97. Arraste o objeto de jogo “menuTitulo” para o primeiro campo e o objeto de jogo, o atual, “menuAjuda” para o segundo, como mostra a Figura A.48;

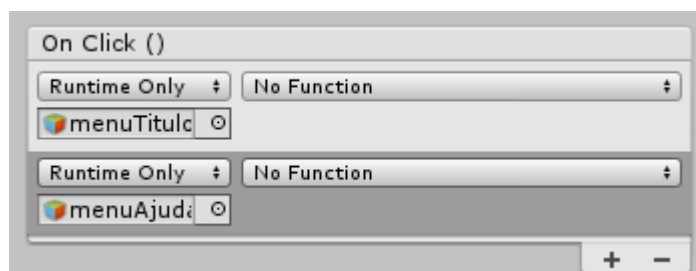


Figura A.48 – Utilização dos objetos de jogo como referência para os eventos ao clicar.

98. Clique em “No Function” para o primeiro evento criado, vá até “GameObject” e então procure por “SetActive (bool)”, como mostra a Figura A.49;

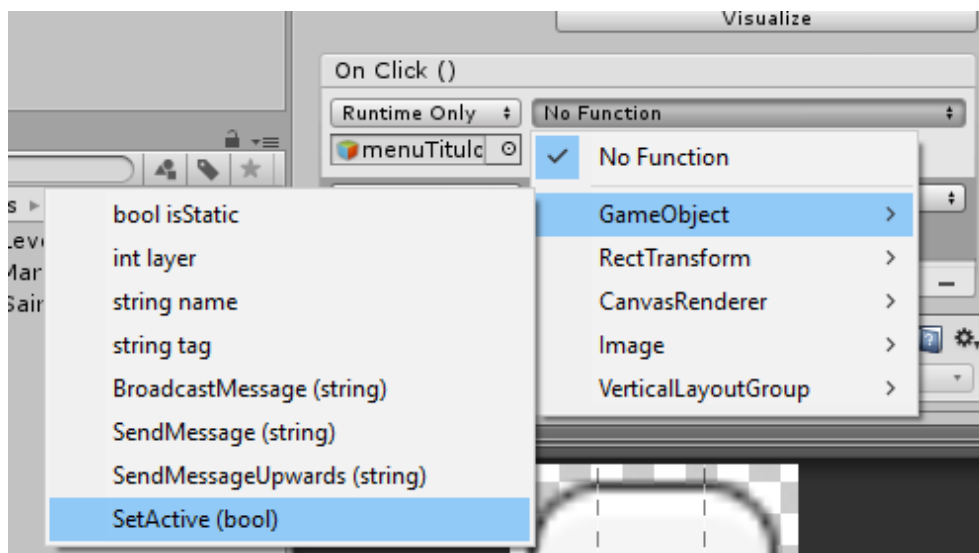


Figura A.49 – Seleção de ativação/desativação de objetos de jogo como evento ao clicar.

99. Refaça o passo 98 para o segundo evento criado;
100. Como o botão voltar faz com que a tela atual saia do “menuAjuda” e vai para o “menuTitulo”, devemos marcar a caixa de seleção para o primeiro evento, e deixar a caixa de seleção, do segundo evento, desmarcada, como mostra a Figura A.50;

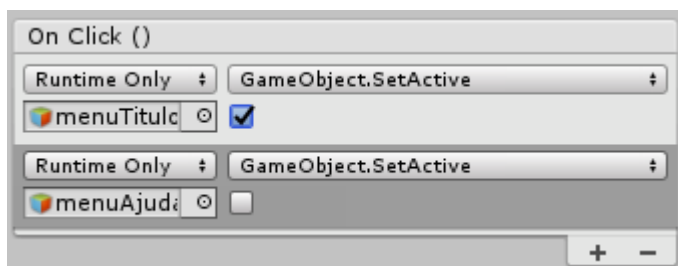


Figura A.50 – Configuração para evento de ativação do objeto de jogo “menuTitulo” e desativação do objeto de jogo “menuAjuda”.

101. Repita do passo 96 ao passo 100, mas, desta vez, para o botão voltar do objeto de jogo “menuSobre” (desta vez, o objeto de jogo atual é o “menuSobre”), como mostra a Figura A.51;

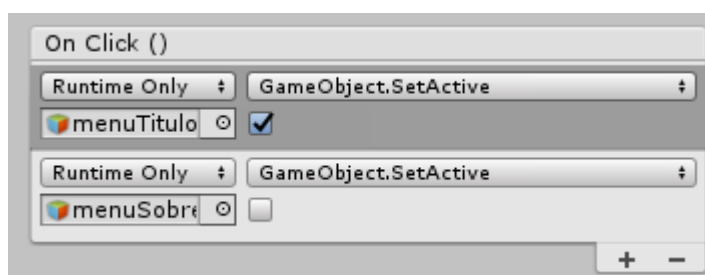


Figura A.51 – Configuração para evento de ativação do objeto de jogo “menuTitulo” e desativação do objeto de jogo “menuSobre”.

102. Repita do passo 96 ao passo 100, mas, desta vez, para os botões de acesso aos objetos “menuAjuda” e “menuSobre”. Para isto, realiza as operações do passo 96 nos botões de “Ajuda” e de “Sobre”. E, desta vez, a seleção na caixa de eventos será ao contrário: o “menuTitulo” ficará vazio e o outro será ativado (“menuAjuda” em um caso e “menuSobre” no outro). Resultado é mostrado nas Figura A.52 e Figura A.53;

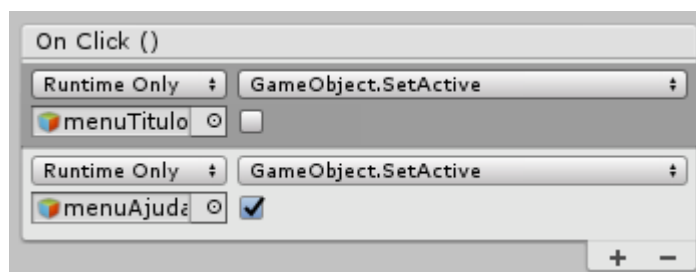


Figura A.52 – Configuração para evento do botão de acesso ao menu “Ajuda” que consiste na ativação do objeto de jogo “menuAjuda” e desativação do objeto de jogo “menuTitulo”.

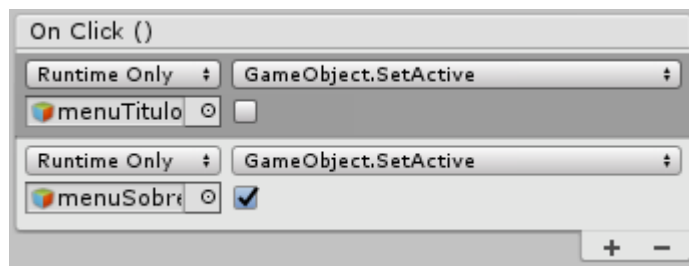


Figura A.53 – Configuração para evento do botão de acesso ao menu “Sobre” que consiste na ativação do objeto de jogo “menuSobre” e desativação do objeto de jogo “menuTitulo”.

103. Repita o passo 29 e crie um *script* em C# com o nome de “CarregaCena”. Feito isto, abra o *script* recém-criado, apague o conteúdo padrão e copie o código do apêndice D para o *script* e salve. Volte para a tela do Unity 3D;
104. Repita os passos 34 e 35 para adicionar o *script* recém-criado ao objeto de jogo “iniciar”, que é o botão para transição entre a cena 0 e a cena 1;
105. Vá até o componente “Button (Script)” e, na configuração “On Click ()”, crie um evento clicando o botão similar a um sinal de “+”;
106. Arraste o componente “Carrega Cena (Script)” para o campo do evento criado e, então, clique em “No Function”, “CarregaCena” e então em “LoadByIndex (int)”, como mostrado na Figura A.54;

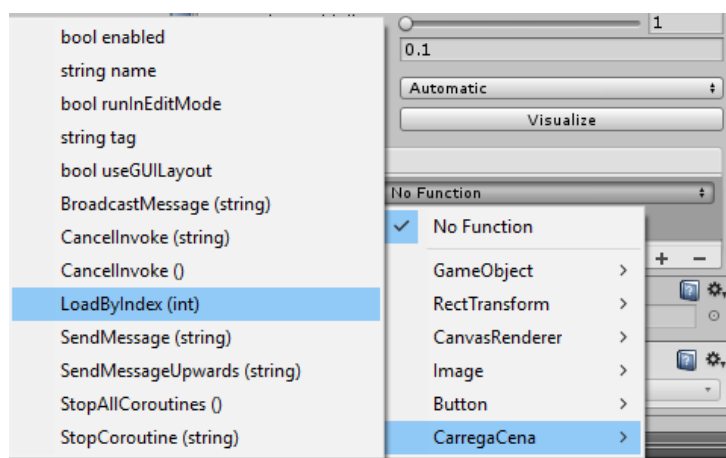


Figura A.54 - Seleção da função “LoadByIndex (int)” para o evento ao clicar.

107. A função “LoadByIndex (int)” requer um argumento de entrada no formato inteiro. Este argumento é o inteiro de identificação das cenas na construção do jogo, como mostrado no passo 45. E, como queremos carregar a cena de inserção de pacientes, iremos preencher o novo campo que aparecerá com o número 1, como mostrado na Figura A.55;

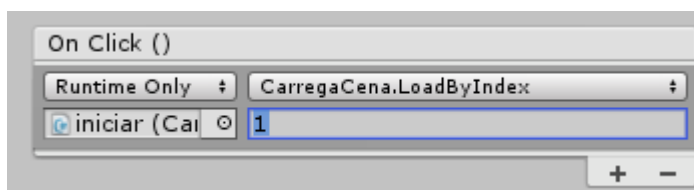


Figura A.55 – Seleção da cena 1 para carregamento ao clique do botão.

108. Salve esta cena e abra a cena “InserirPaciente”, na pasta “Cenas”;
109. Repita os passos do 47 ao 52, para criar um painel e, nomeie-o como “inserirPaciente”.
Salve as cenas;
110. Dentro de um projeto, podemos aproveitar situações já desenvolvidas em outras cenas. Iremos utilizar deste recurso para otimizar o desenvolvimento do jogo. Vá até a cena “MenuPrincipal” e copie (através dos comandos básicos ou através do clique do botão direito do mouse sobre o objeto de interesse) os seguintes objetos de jogo para a cena “InserirPaciente”:
 - a. LevelLoader;
 - b. carregando... (lembre-se que este painel deste estar dentro do painel *Canvas* criado no passo 109);
111. Na janela *Hierarchy*, deixe ativado somente o objeto “inserirPaciente” (dentro do painel *Canvas*), como mostra a Figura A.56;

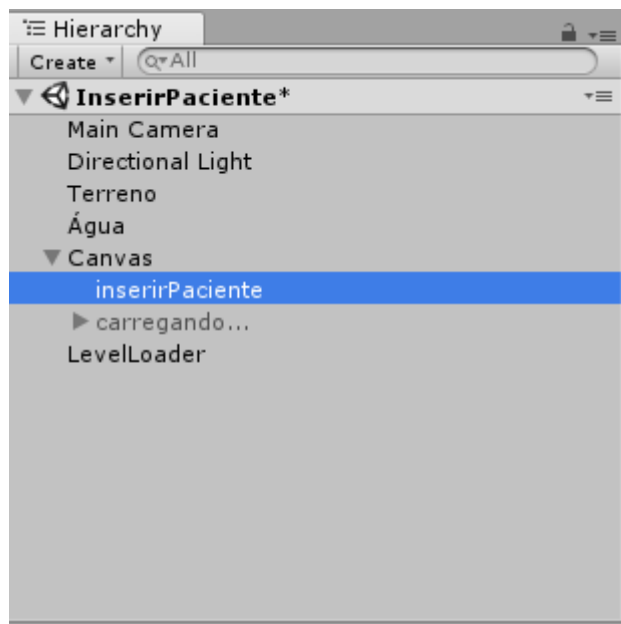


Figura A.56 – Janela *Hierarchy* mostrando apenas o painel “inserirPaciente” ativado.

112. Crie uma caixa de texto dentro do objeto “inserirPaciente” conforme a Figura A.57. Nomeie-a como “tituloInserirPaciente”. E, através do componente “Text (Script)”, mude o texto de apresentação para “Inserir novo usuário”;

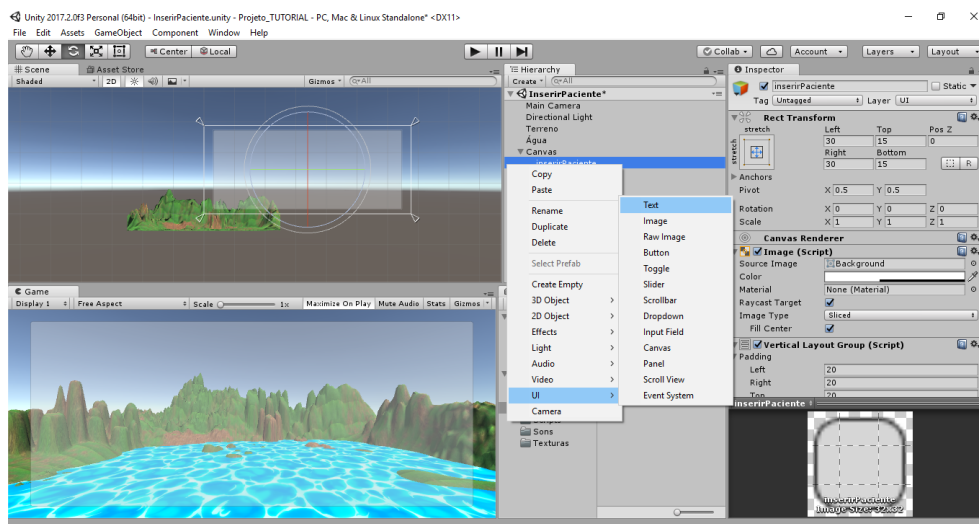


Figura A.57 – Criação de um objeto de jogo do tipo texto como filho de um objeto de jogo do tipo painel.

113. Configure o tamanho da fonte para 20 e coloque em negrito, no estilo de fonte. Em “Paragraph”, na configuração “Alignment”, escolha centralizar horizontalmente e verticalmente;
114. Repita o passo 112 para a criação de uma nova caixa de texto (não é necessário modificar o nome ainda). Em seus componentes, modifique as configurações para que fiquem idênticas a da Figura A.58;

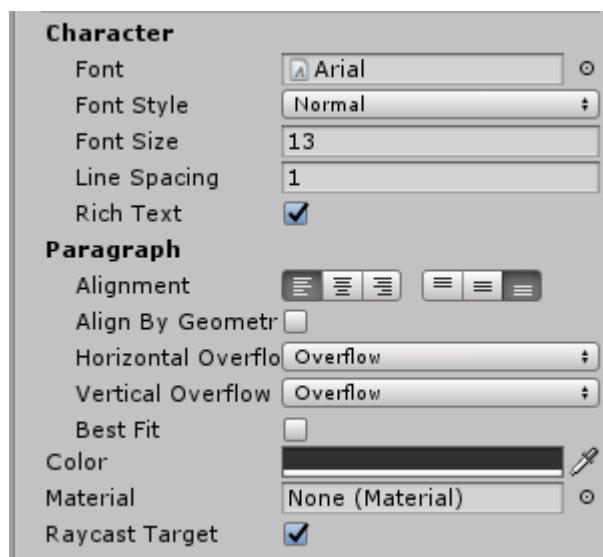


Figura A.58 – Componente “Text (Script)” do objeto de jogo do tipo texto.

115. Conforme a Figura A.59, na janela *Hierarchy*, clique com o botão direito no objeto “inserePaciente” e então clique em “UI” e, por fim, clique em “Input Field” e crie uma caixa de inserção de texto;

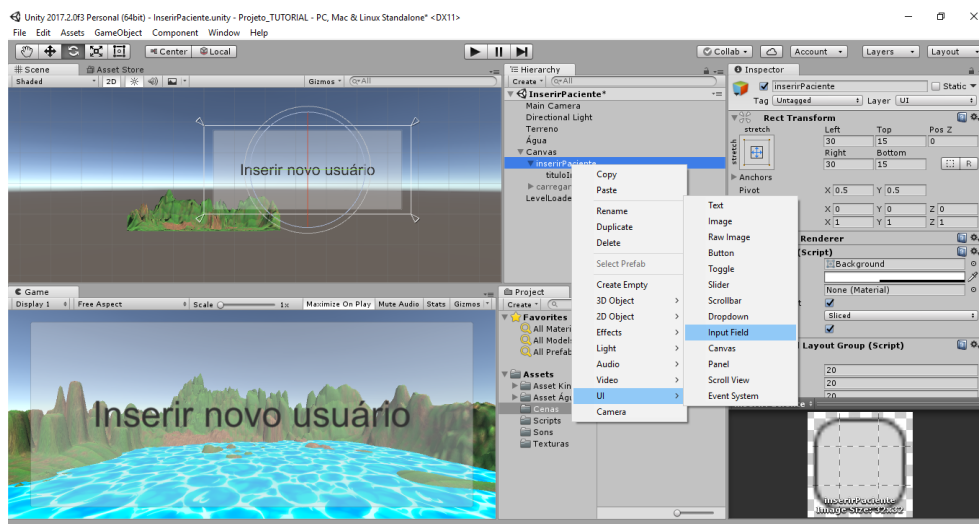


Figura A.59 – Criação de um objeto de jogo do tipo “campo de entrada” para inserção de informações por meio do usuário durante o jogo.

116. Nos elementos filhos do recém-criado “Input Field”, temos dois elementos de texto. Para o elemento de nome “Placeholder”, as configurações do componente “Text (Script)” devem ficar como na Figura A.60.

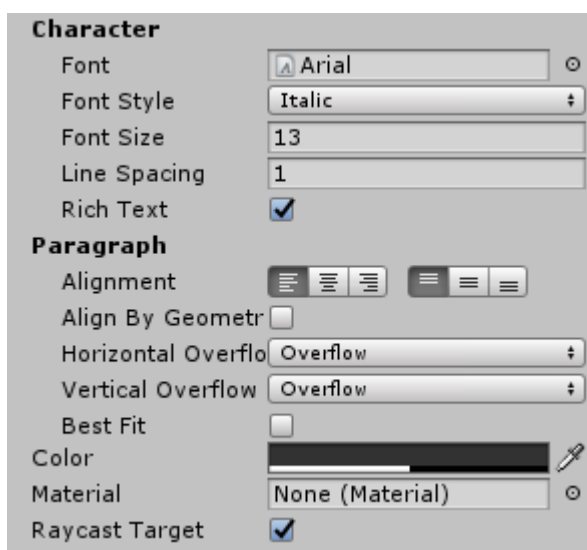


Figura A.60 – Componente “Text (Script)” do objeto de jogo “Placeholder”.

E para o componente de nome “Text”, ele deve ficar como mostra a Figura A.61;

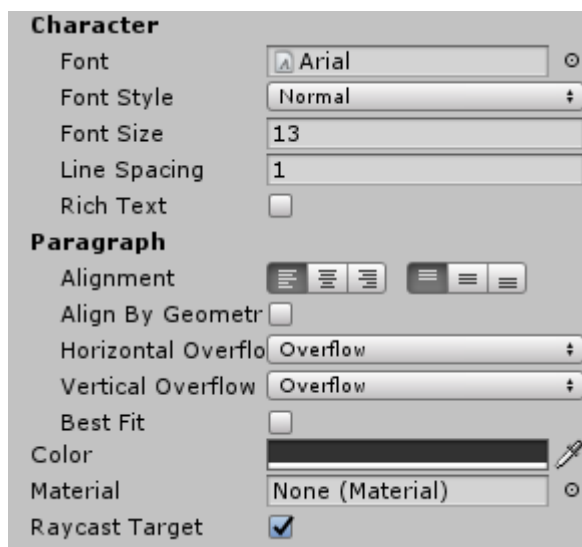


Figura A.61 – Componente “Text (Script)” do objeto de jogo “Text”.

117. Selecione os dois objetos de jogo recém-criados e duplique-os por mais 7 vezes, de forma que, no final, obtenha-se 8 objetos iguais, tendo uma janela *Hierarchy* como mostrada na Figura A.62;

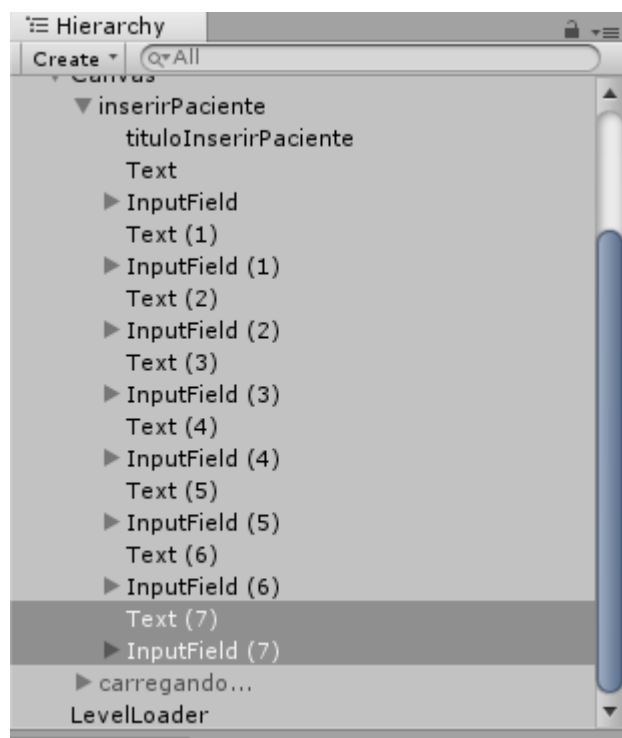


Figura A.62 – Janela *Hierarchy* após a criação dos outros campos de inserção e seus títulos.

118. Se após o passo anterior o jogo parecer deslocado (ou não estiver aparecendo todos os objetos) na janela *Game*, vá até o componente “Vertical Layout Group” no objeto “inserirPaciente” e, em “Spacing”, reduza o valor até que o jogo fique esteticamente bom. Estes valores são determinados pela tela utilizada. Para ter uma melhor ideia do jogo, clique no canto superior direito da janela *Game* e clique em “Maximize”, como mostra a Figura A.63. (Para voltar a edição, faça o mesmo procedimento);



Figura A.63 – Mecanismo de maximização de uma das guias dentro do software Unity 3D.

119. Dê nome aos objetos de jogo criados para que fiquem como mostrado na Figura A.64;

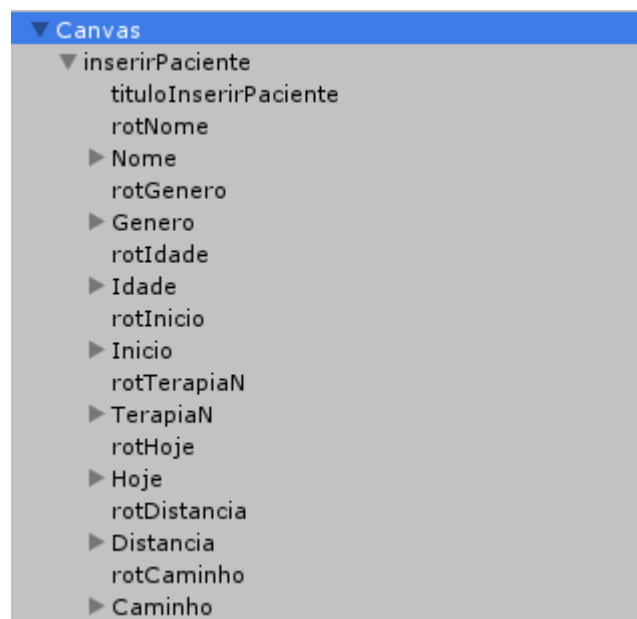


Figura A.64 – Objetos de jogo após serem renomeados.

120. E então, nomeie os objetos de texto com nome “Text” dentro de cada um dos objetos do tipo “Input Field” de forma a ficar o nome do objeto pai, acrescentado de “Texto” sem espaços, como mostra a Figura A.65;

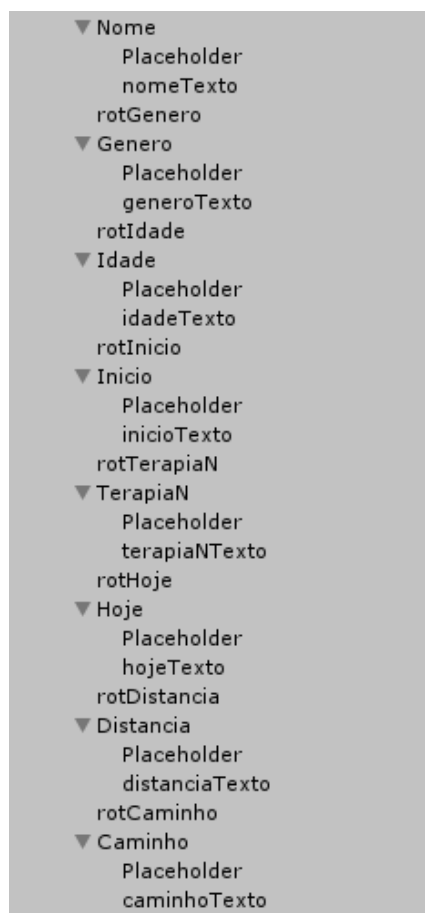


Figura A.65 – Nomeação dos objetos filhos.

121. Nos objetos de jogo de texto, edite o texto de cada um deles, como mostra a Figura A.66, no componente “Text (Script)” dos objetos de jogo com a *tag* “rot”, de acordo com o nome de cada um deles: Nome, Gênero, Idade, Data de início de tratamento, Número da terapia atual, Data de hoje, Distância entre os obstáculos (entre 10 e 100), Diretório para salvar o relatório;

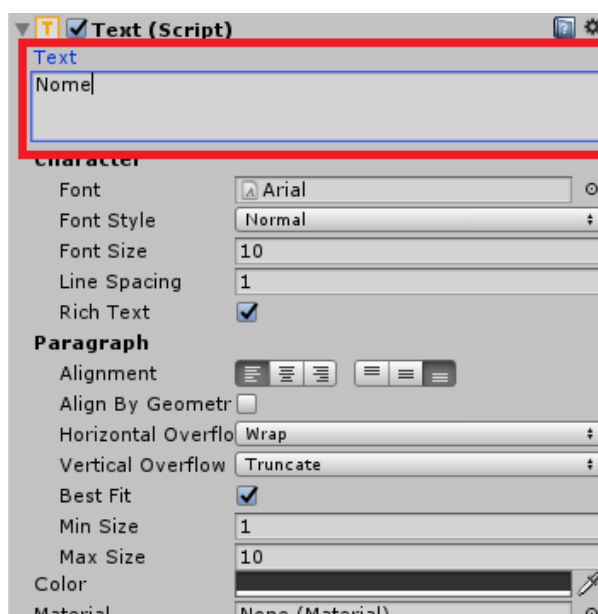


Figura A.66 – Campo de modificação de texto destacado.

E, então, a tela ficará da seguinte forma, como mostra a Figura A.67;

Figura A.67 – Visão em momento intermediário do desenvolvimento da cena “InserirPaciente”.

122. Agora, clique com o botão direito no objeto de jogo “inserirPaciente” e, então em “UI” e, por fim, insira um botão clicando em “Button” e nomeie-o como “SalvarIniciar”;
123. Modifique o texto de apresentação do mesmo, em seu objeto filho, no componente “Text (Script)” para “Salvar dados e iniciar o JOGO!”;
124. Nas configurações “Paragraph”, marque a opção “Best Fit” e selecione o valor mínimo para 10 e o valor máximo para 40;
125. Crie um objeto vazio, como explicado no passo 38 e dê o nome a ele de “Usuario”;
126. Dentro da pasta “Scripts”, crie um novo *script*, como mostrado no passo 29 e nomeie-o como “SalvarArquivo”. Abra o *script* recém-criado, apague o conteúdo padrão e copie o código presente no apêndice M;
127. Vá até o objeto de jogo “Usuario” e selecione-o. Na janela *Inspector*, clique em “Add Component” e, no mecanismo de busca, pesquise por “Salvar Arquivo” e adicione o mesmo ao objeto. E então, o componente ficará como mostra a Figura A.68;

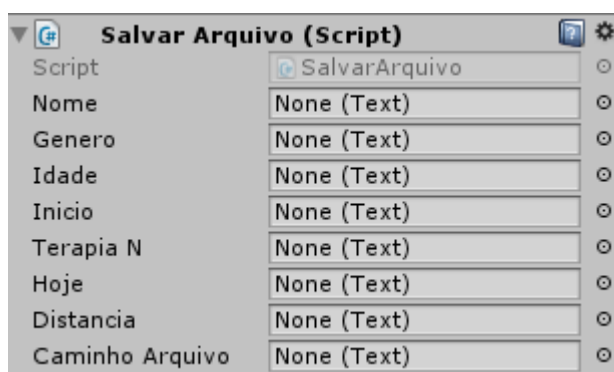


Figura A.68 – Componente “Salvar Arquivo (Script)” dentro do objeto de jogo “Usuario”.

128. Para completar as variáveis, devemos arrastar os objetos de jogo referente as mesmas. Os objetos destas variáveis são os filhos dos objetos do tipo “Input Field” nomeados no passo 120. Clique e arraste para cada um dos campos e, o resultado final, ficará como mostra a Figura A.69;



Figura A.69 – Componente “Salvar Arquivo (Script)” com variáveis totalmente definidas.

129. Por fim, selecione o botão “SalvarIniciar” e adicione o *script* “CarregaCena” como componente deste objeto como feito nos passos 34 e 35;
130. Vá até o componente “Button (Script)” e, na configuração “On Click ()”, crie um evento clicando o botão similar a um sinal de “+”;
131. Arraste o componente “Carrega Cena (Script)” para o campo do evento criado e, então, clique em “No Function”, “CarregaCena” e então em “LoadByIndex (int)”, como mostrado na Figura A.70;

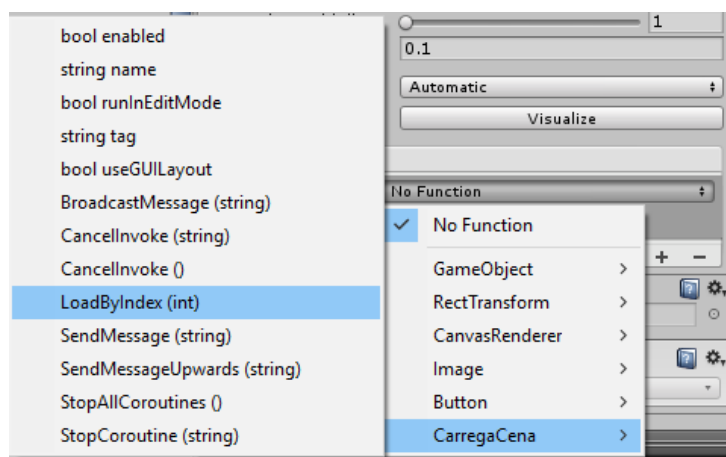


Figura A.70 - Seleção da função “LoadByIndex (int)” para o evento ao clicar.

132. A função “LoadByIndex (int)” requer um argumento de entrada no formato inteiro. Este argumento é o inteiro de identificação das cenas na construção do jogo, como mostrado no passo 42. E, como queremos carregar a cena de inserção de pacientes, iremos preencher o novo campo que aparecerá com o número 2, como mostrado na Figura A.71;

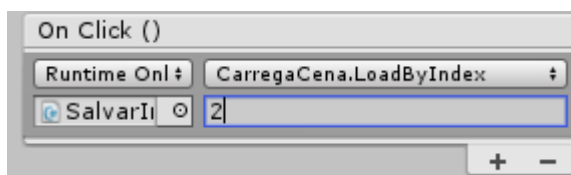


Figura A.71 - Seleção da cena 2 como parâmetro de entrada da função “LoadByIndex (int)”.

133. Repita o passo 130;
134. Arraste o objeto de jogo “Usuario” para o campo gerado pelo novo evento;

135. Clique em “No Function” e então em “SalvarArquivo”. Escolha a opção “SaveButtonOnClick()” clicando nela, como mostrado na Figura A.72;

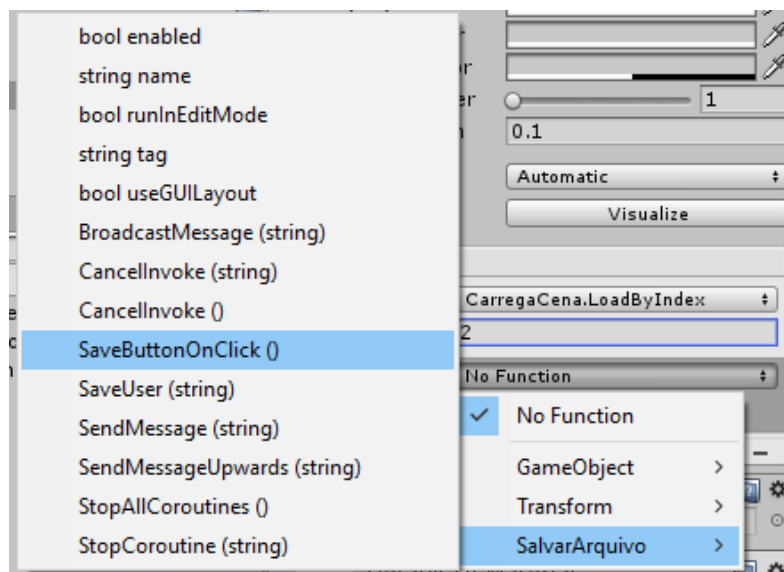


Figura A.72 - Seleção da função “SaveButtonOnClick ()” para o evento ao clicar.

136. Repita os passos 86 ao 89 para preencher as variáveis do *script* “Level Loader”, presente no objeto de jogo “LevelLoader” copiado da cena anterior no passo 110. Neste caso, a variável “Main Menu” é o painel “InserirPaciente”;
137. Salve a cena “InserirPaciente” e abra a cena “JogoPrincipal”;
138. Repita o passo 5 e crie uma nova pasta chamada “Pré-Fabricados”;
139. A ponte do jogo desenvolvido neste trabalho foi feita no software SolidWorks® e, então, importada para o software Unity 3D, como apresentado no corpo do trabalho. Desta forma, este objeto será disponibilizado via link mostrado abaixo para a utilização gratuita. Faça o download e salve todo o conteúdo na pasta criada “Pré-fabricados”, como mostra a Figura A.73;

https://drive.google.com/open?id=1SIYW57k8l6nJM8_h_DpcptxG3SR9kDd5 – Acesso em 26/01/2018

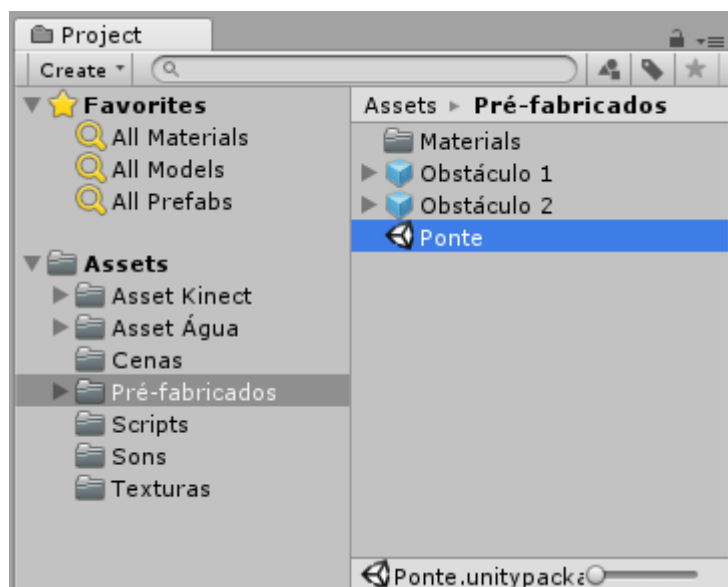


Figura A.73 – Visualização dos componentes da pasta “Pré-fabricados”.

140. Clique duas vezes no arquivo “Ponte”, selecionado na Figura A.70, para importar o *package* feito para o modelo pré-fabricado “Ponte”. Desta vez, importe todos os arquivos presentes no *package*. Após importado, certifique de que estão na pasta “Pré-fabricados” e, caso necessário, você pode excluir a pasta padrão do mesmo “Prefabs” e a pasta “Materiais”, dentro da pasta “Prefabs”. Arraste o objeto “PecaPonte” da janela *Project* para a janela *Hierarchy*. Renomeie o objeto recém-criado para “Ponte”. A hierarquia ficará como mostrado na Figura A.74;

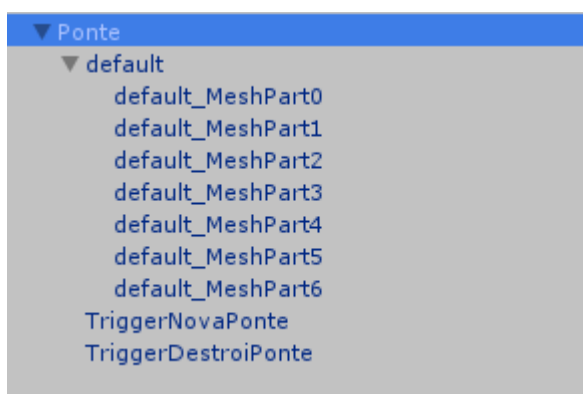


Figura A.74 – Hierarquia do objeto de jogo “Ponte”.

141. Para selecionarmos os materiais do objeto recém-adicionado à hierarquia do jogo, abra a pasta “Materiais” que está dentro da pasta “Pré-fabricados”, obtida no passo 139. Selecione o arquivo de material “medwood” clicando sobre ele (ele é representado pelo ícone de esfera). Para adicionar uma textura ao material, arraste o arquivo de textura de mesmo nome “medwood” (similar a uma imagem) para o local “Albedo” indicado em azul na janela *Inspector* como mostra a Figura A.75;

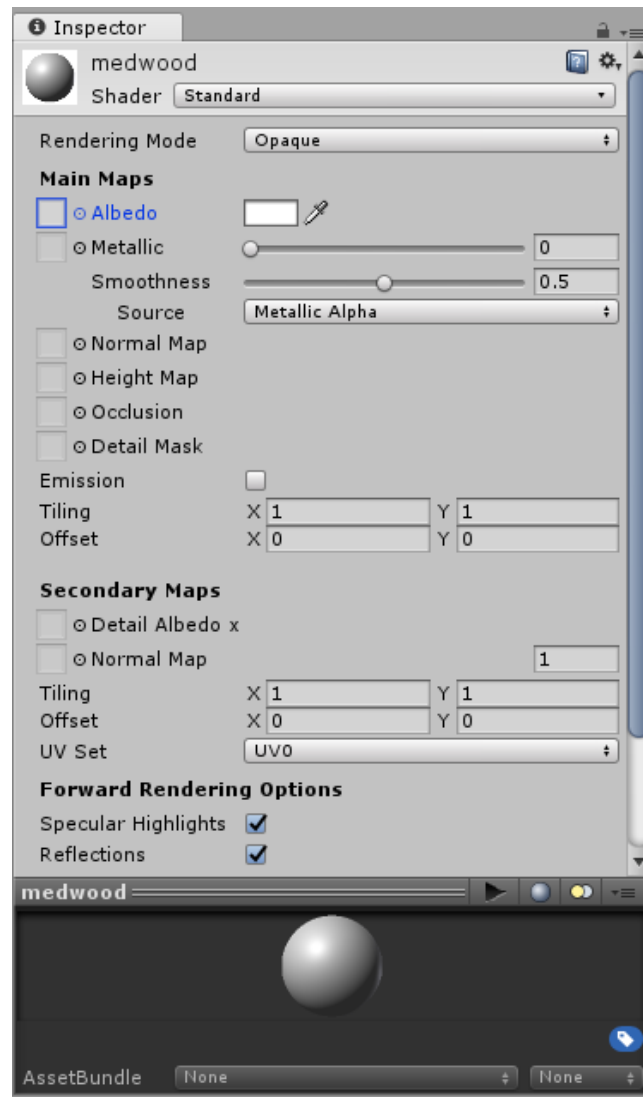


Figura A.75 – Visão da janela *Inspector* de um material.

142. Repita o passo 141 para os materiais “oak”, “Estrela – Particula” e “Obstaculo”;
143. Arraste os arquivos de materiais, presentes na janela *Project* para seus componentes na janela *Hierarchy* (filhos do objeto filho do objeto “Ponte”, o objeto “default”), da seguinte forma. Eles estão dispostos como mostra a Figura A.76:
 - a. default_MeshPart0 -> medwood;
 - b. default_MeshPart1 -> medwood;
 - c. default_MeshPart2 -> oak;
 - d. default_MeshPart3 -> oak;
 - e. default_MeshPart4 -> oak;
 - f. default_MeshPart5 -> oak;
 - g. default_MeshPart6 -> oak;

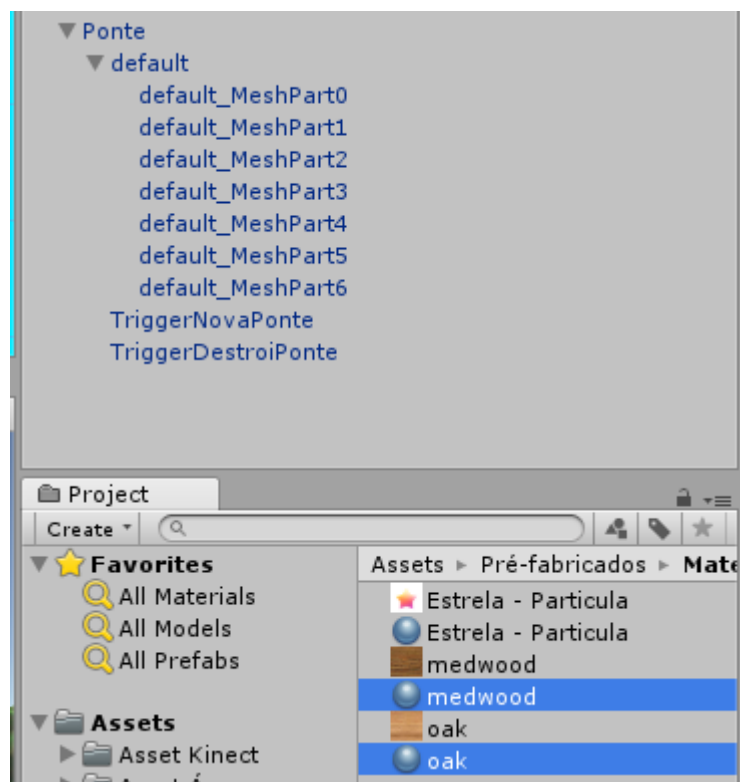


Figura A.76 – Disposição de alguns elementos dentro da interface da Unity 3D.

144. Vá até a pasta “Asset Kinect” e, acesse a pasta “KinectDemos”. Em “AvatarsDemo”, abra a cena “KinectAvatarsDemo1” como mostra a Figura A.77;

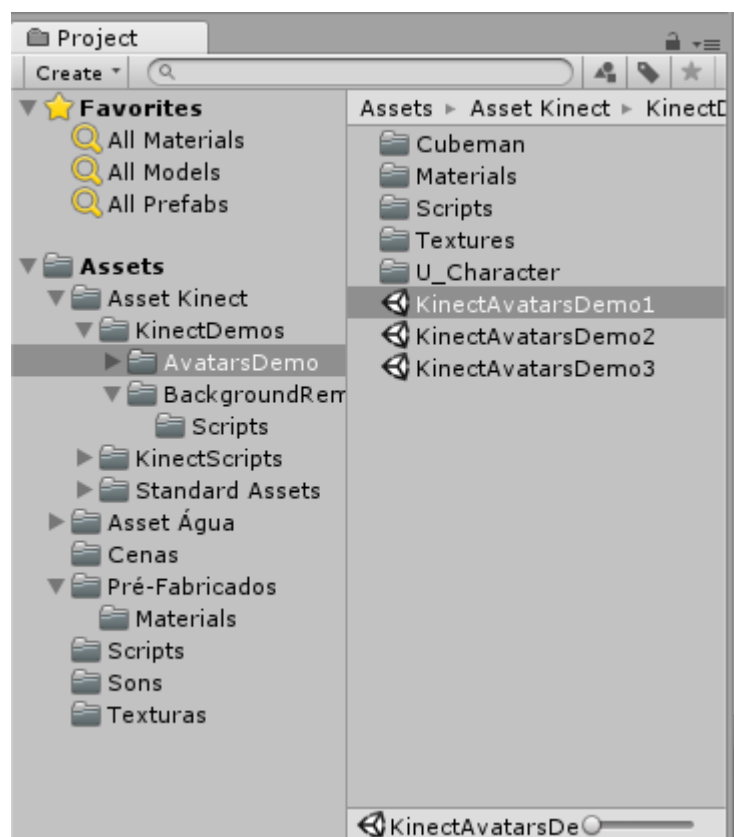


Figura A.77 – Caminho para a cena “KinectAvatarsDemo1” do package “Kinect v2 Examples MS_SDK” importado anteriormente.

145. Copie os dois objetos de jogo mostrados na Figura A.78: O “U_CharacterBack” e o objeto “KinectController”. Feito isto, volte para a cena “JogoPrincipal” na pasta “Cenas” e cole os objetos copiados na janela *Hierarchy*;

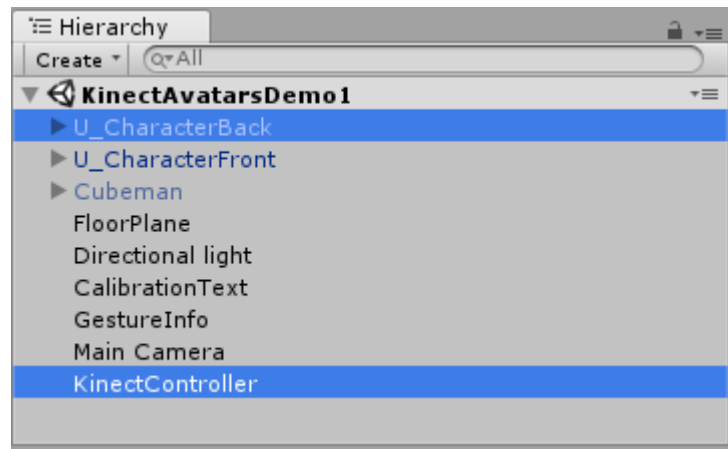


Figura A.78 – Objetos de jogo necessários para o jogo.

146. Renomeie o objeto “U_CharacterBack” para “Avatar” e, em seu componente *Transform*, modifique-o de forma que o mesmo fique idêntico ao da Figura A.79;

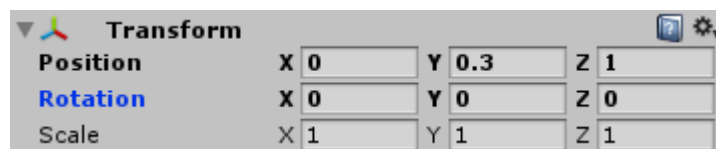


Figura A.79 – Componente *Transform* do objeto de jogo “Avatar”.

147. Arraste o objeto de jogo “Main Camera” para que o mesmo se torne um objeto filho do objeto recém-criado “Avatar”. O resultado final deve ser como mostrado na Figura A.80;

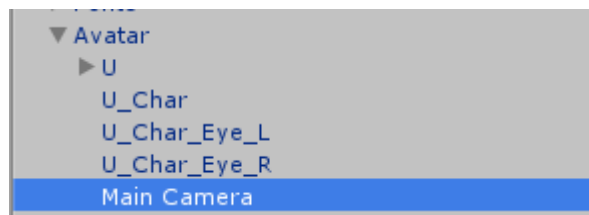


Figura A.80 – Visão de parte da hierarquia do objeto de jogo “Avatar”.

148. Com o objeto de jogo “Main Camera” selecionado, modifique seu componente *Transform* para que fique idêntico ao da Figura A.81;

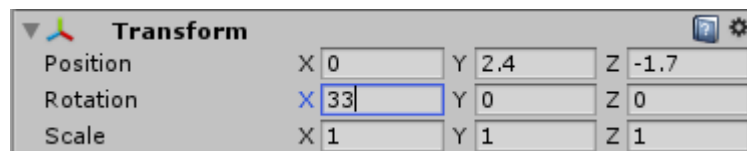


Figura A.81 – Componente *Transform* do objeto de representação da câmera, “Main Camera”.

149. Na pasta “Pré-fabricados” na janela *Project*, com o mouse, arraste o objeto de jogo “Obstáculo 1” para a janela *Hierarchy*;

150. Na pasta “Materiais”, dentro da pasta “Pré-fabricados”, arraste o material “Obstaculo” (representado pela esfera com nome “Obstaculo”) e solte na janela *Project* em cima do objeto de jogo “Obstáculo 1”;
151. Repita o passo 149 e 150 para o objeto de jogo “Obstáculo 2”;
152. Identifique os objetos do “Avatar” como mostrado na Figura A.82 e, então, na janela *Inspector*, desative os componentes de colisão (Collider) presente neles, para que estes não influenciem nas informações obtidas para os gatilhos utilizados no jogo;



Figura A.82 – Parte da hierarquia do objeto de jogo “Avatar”.

153. Dentro da pasta “Scripts”, crie um novo *script*, como mostrado no passo 29 e nomeie-o como “VariacaoVelocidadeAvatar”. Abra o *script* recém-criado, apague o conteúdo padrão e copie o código presente no apêndice P;
154. Vá até o objeto de jogo “joint_ToeLT” e selecione-o. Na janela *Inspector*, clique em “Add Component” e, no mecanismo de busca, pesquise por “Variacao Velocidade Avatar” e adicione o mesmo ao objeto.
155. Repita o passo 154 para o objeto de jogo “joint_ToeRT” mas, neste caso, apenas precisamos de informações para outros *scripts* então, desative o mesmo, como mostra a Figura A.83;

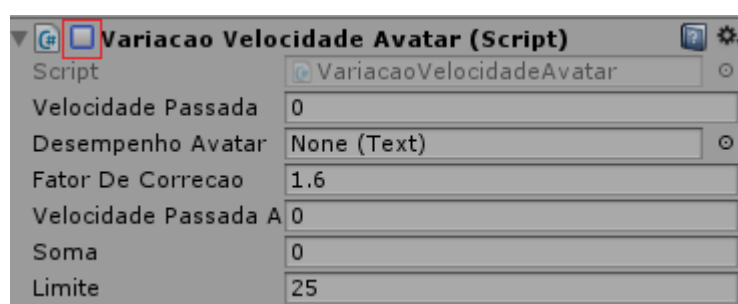


Figura A.83 – Componente desativado, sendo utilizado apenas para obtenção de informações para outros objetos.

156. No objeto “U_Char”, filho do objeto “Avatar”, vá em “Add Component” e procure por “Capsule Collider” e adicione o mesmo ao objeto, como mostra a Figura A.84;

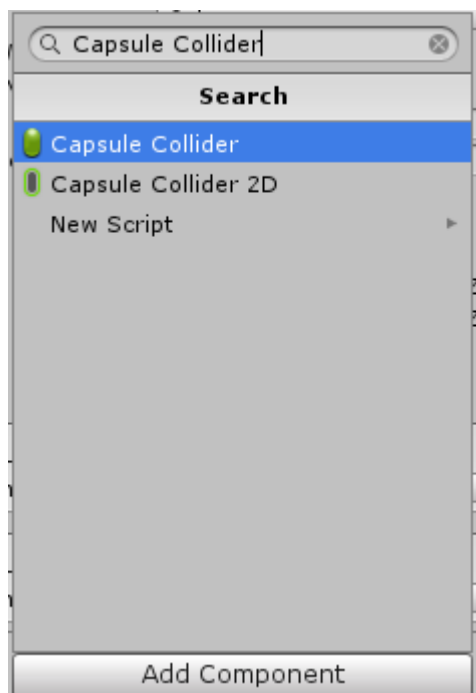


Figura A.84 – Inserção do componente “Capsule Collider” em um objeto de jogo.

157. Agora, configure este componente para fique como o mostrado na Figura A.85;

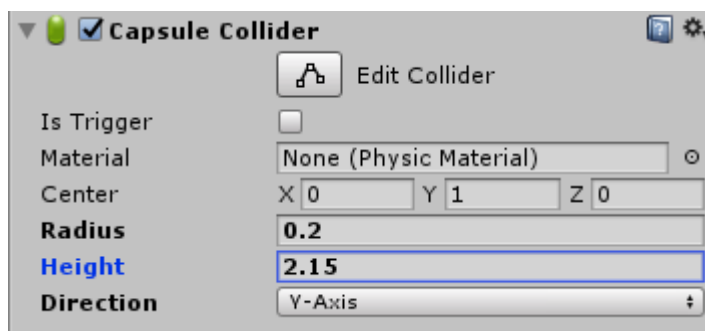


Figura A.85 – Configuração do componente “Capsule Collider”.

158. No objeto de jogo “Avatar” ainda, configure o componente “Rigidbody” para que o mesmo fique igual ao da Figura A.86;

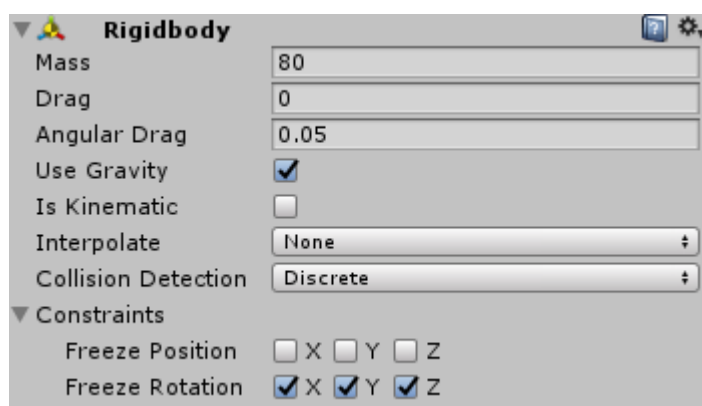


Figura A.86 – Configuração do componente “Rigidbody”.

159. Agora, vá até o objeto “Ponte” e, seu objeto filho “TriggerNovaPonte” e selecione-o, como mostra a Figura A.87;

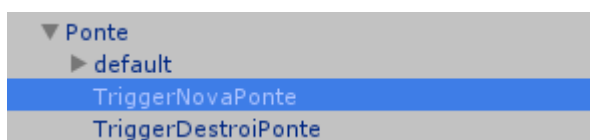


Figura A.87 – Destaque do objeto de jogo “TriggerNovaPonte”.

160. Note que no componente “Cria Ponte (Script)” está com um campo “Ponte Prefab”. Arraste o objeto de jogo “Ponte” para o campo à direita da configuração “Ponte Prefab”;
161. Nos objetos “Ponte”, “Obstáculo 1” e “Obstáculo 2”, certifique-se de que o componente “Velocidade Ponte (Script)” está ativado, como mostra a Figura A.88;

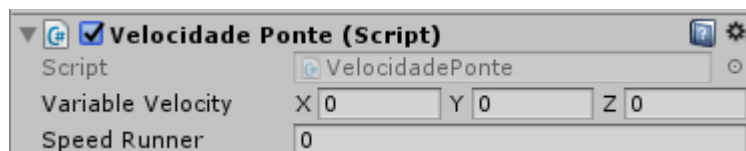


Figura A.88 – Visualização do componente de jogo “Velocidade Ponte (Script)” ativado em um objeto de jogo.

162. Dentro da pasta “Scripts”, crie um novo *script*, como mostrado no passo 29 e nomeie-o como “PosicaoAleatoriaObs”. Abra o *script* recém-criado, apague o conteúdo padrão e copie o código presente no apêndice J;

163. No objeto de jogo “Obstáculo 1”, em seu objeto filho “TriggerObstaculo”, no componente “Nothing Select”, há um campo para a inserção de um novo *script*. Vá até a pasta “Scripts” e arraste o *script* “PosicaoAleatoriaObs” para o campo em questão, como mostra a Figura A.89;

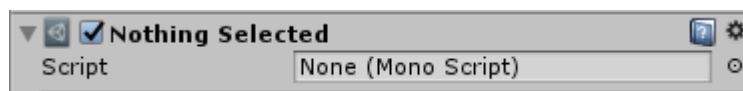


Figura A.89 – Campo para inserção de um *script* em um objeto de jogo importado externamente.

164. Repita o passo 163 para o objeto de jogo “Obstáculos 2”;
165. Dentro da pasta “Scripts”, crie um novo *script*, como mostrado no passo 29 e nomeie-o como “DestroiObstaculo”. Abra o *script* recém-criado e copie o código presente no apêndice F;
166. No objeto de jogo “Obstáculo 1”, no objeto filho “Particle System”, vá até o componente de textura (neste caso, com o nome de “Estrela – Particula”) e expanda-o. Em “Emissive Color”, certifique-se que a cor seja branca, conforme mostra a Figura A.90;

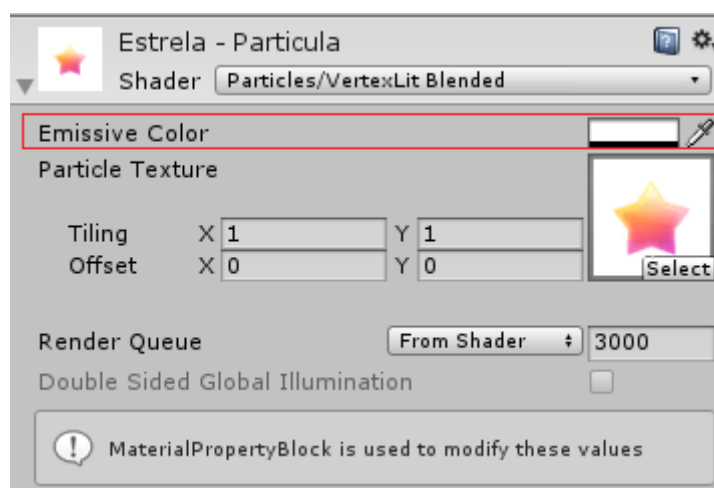


Figura A.90 – Textura para as partículas da explosão.

167. Repita o passo 166 para o objeto de jogo “Obstáculo 2”;
168. No objeto de jogo “Obstáculo 1”, em seu componente filho “TriggerObstaculo”, vá até o componente “Posicao Aleatoria Obs (Script)” para completar as variáveis do *script* da seguinte maneira;
- Z Position: não altere nada;
 - Obstacle: objeto de jogo “Obstáculo 2”; e
 - Explosao: objeto de jogo “Particle System” do objeto de jogo “Obstáculo 1”.

Para completar as variáveis como mostrado acima, faça como nos passos 87, 88 e 89;

169. Repita o procedimento anterior para o objeto de jogo “Obstáculo 2”, fazendo a alternância no passo 168.b para objeto de jogo “Obstáculo 1” e, no passo 168.c para objeto de jogo “Obstáculo 2”;
170. Agora, devemos atualizar os objetos pré-fabricados. Para isto, arraste-os da janela *Hierarchy* para a janela *Project*, a pasta “Pré-fabricados” e solte-os em cima de seus respectivos objetos de origem (por exemplo, arraste o objeto de jogo “Obstáculo 1” e solte em cima do arquivo “Obstáculo 1”, como mostra a Figura A.91);

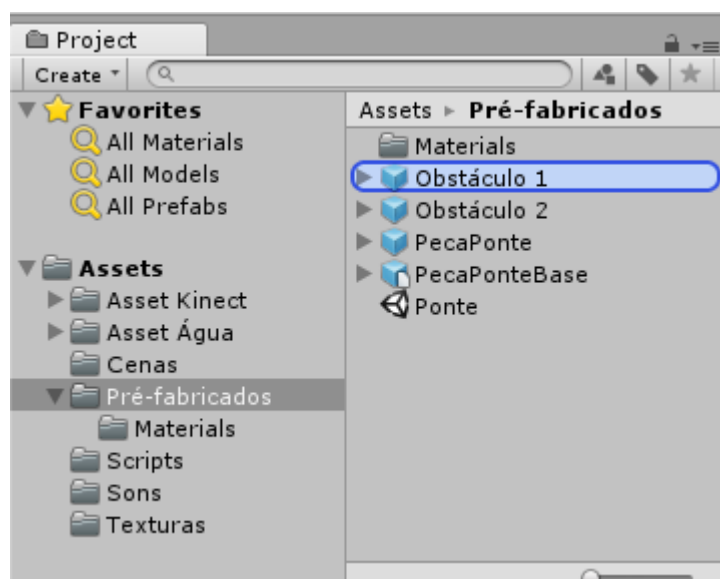


Figura A.91 – Atualização dos objetos pré-fabricados.

171. Repita o passo 170 para o arquivo da pasta “Pré-fabricados” “Obstáculo 2”;
172. Exclua o objeto de jogo “Obstáculo 2” da janela *Hierarchy*;
173. Crie um painel *Canvas* clicando, na janela *Hierarchy*, na seleção “Create”, como mostra a Figura A.92;

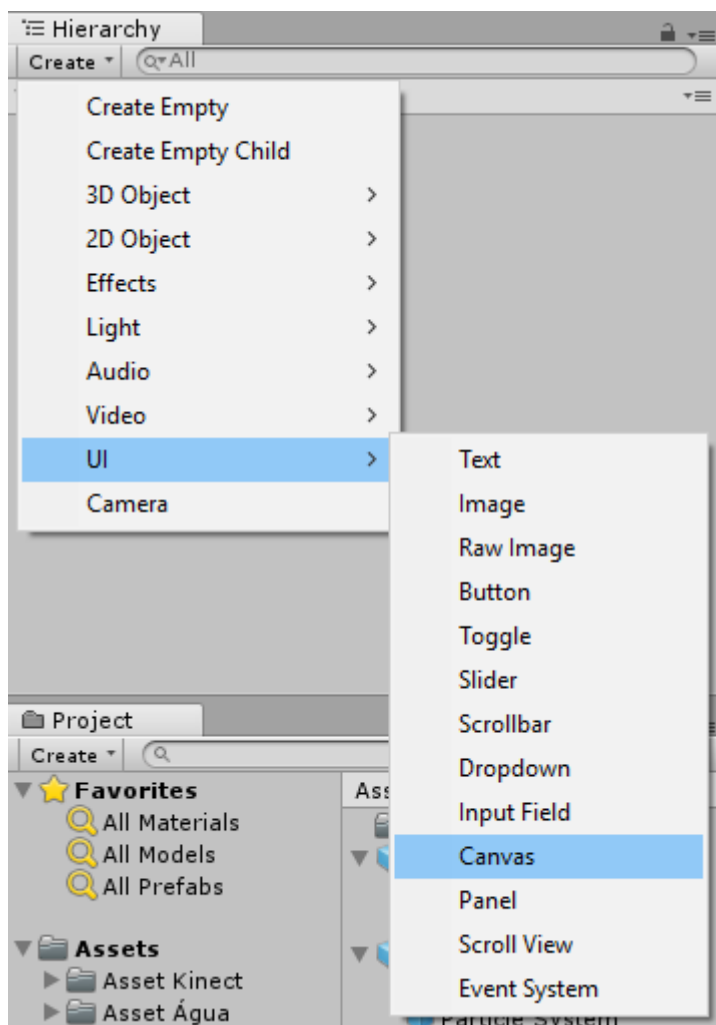


Figura A.92 – Criação de um objeto de jogo do tipo “Canvas”.

174. Na janela *Inspector* do objeto de jogo “Canvas”, configure o componente “Canvas Scaler (Script)” para que fique idêntico ao da Figura A.93;

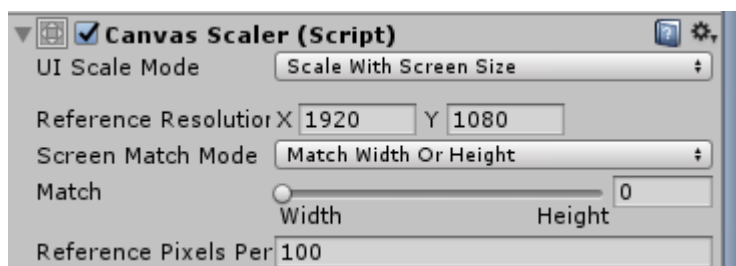


Figura A.93 – Configuração da escala do painel *Canvas* criado em relação a tela de exibição do jogo.

175. Clique com o botão direito sobre o objeto “Canvas” recém-criado e, então, clique sobre “UI” e, selecione “Panel” para criar um painel. Nomeie-o como “informacoesDeJogo”;
176. Com o objeto “informacoesDeJogo” selecionado, vá até o componente *Rect Transform* do mesmo e configure-o para que o mesmo fique idêntico ao mostrado na Figura A.94;

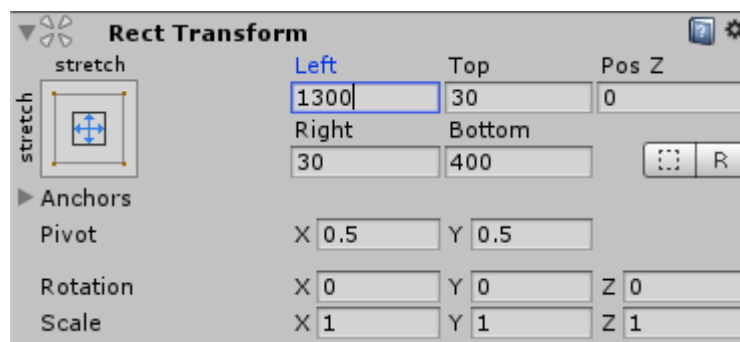


Figura A.94 – Configuração do componente *Rect Transform* do objeto de jogo “informacoesDeJogo”.

177. Repita o passo 51 para adicionar o componente “Vertical Layout Group” ao painel em questão e, então, configure-o para as informações mostradas na Figura A.95;

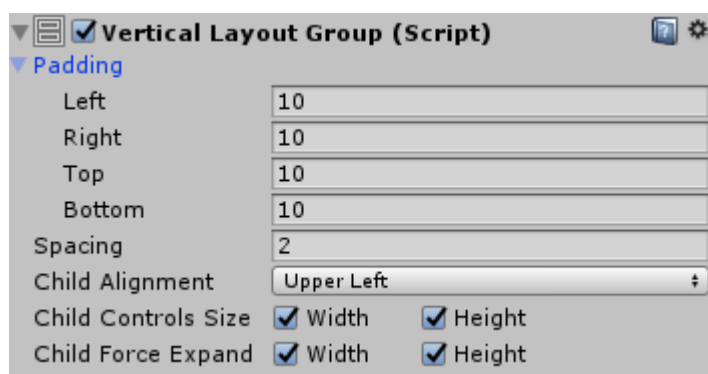


Figura A.95 – Configuração do componente de jogo “Vertical Layout Group (Script)”.

178. Clique com o botão direito no objeto de jogo “informacoesDeJogo”, selecione “UI” e crie 3 caixas de texto (“Text”) e um botão (“Button”), exatamente nesta ordem;
179. Dê os nomes aos objetos, como mostrado na Figura A.96;

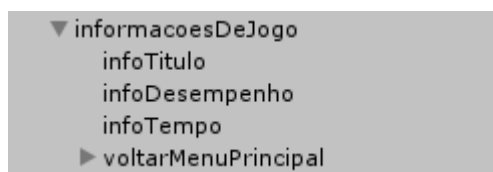


Figura A.96 – Hierarquia do painel *Canvas*.

180. Selecione o objeto “infoTitulo” e, no componente “Text (Script)”, edite o texto padrão “New Text” para o “Informações”. Em “Font Style”, marque a opção “Bold”. Em “Paragraph”, marque as opções de centralização vertical e horizontal e também a opção “Best Fit” no mesmo componente e defina os valores de mínimo para 20 e de máximo para 40;
181. Selecione o objeto “infoDesempenho” e, em “Paragraph”, marque as opções de centralização vertical e horizontal e também a opção “Best Fit” no mesmo componente e defina os valores de mínimo para 5 e de máximo para 30;
182. Repita o passo 181 para o objeto “infoTempo”;

183. Repita o passo 58 para configuração do botão “voltarMenuPrincipal” e edite seu texto padrão para “Voltar para o Menu”. Marque a opção “Best Fit” e defina os valores de mínimo para 10 e de máximo para 40;
184. Crie um objeto vazio, clicando em “Create” na janela *Hierarchy* e, então em “Create Empty”. Dê o nome ao objeto criado de “Tempo”;
185. Repita o passo 29 para criar um *script* C# (dentro da pasta “Scripts”) e nomeie-o como “TempoExecucao”. Feito isto, abra o *script* recém-criado, apague o conteúdo padrão e copie o código do apêndice O para o *script* e salve. Volte para a tela do Unity 3D;
186. Repita o passo 34 e 35 para adicionar o *script* “TempoExecucao” como um componente do objeto de jogo “Tempo”;
187. No componente “Tempo Execucao (Script)” do objeto de jogo “Tempo”, complete a variável “Execution” arrastando o objeto de jogo “infoTempo” para o campo à direita da mesma;
188. Vá até o objeto de jogo “joint_ToeLT” dentro do objeto de jogo “Avatar” para completar a variável “Desempenho Avatar” arrastando o objeto de jogo “infoDesempenho” para o campo a direita da mesma;
189. Selecione agora o objeto de jogo “voltarMenuPrincipal” e, conforme o passo 93, crie um evento;
190. Clique em “Add Component” e, usando o campo de busca, procure por “Carrega Cena” e adicione-o ao objeto;
191. Arraste o componente “Carrega Cena” para o evento criado e, selecione a opção mostrada na Figura A.97, “LoadByIndex (int)”. No campo que surgiu, preencha com 0, uma vez que este botão irá carregar a cena 0;

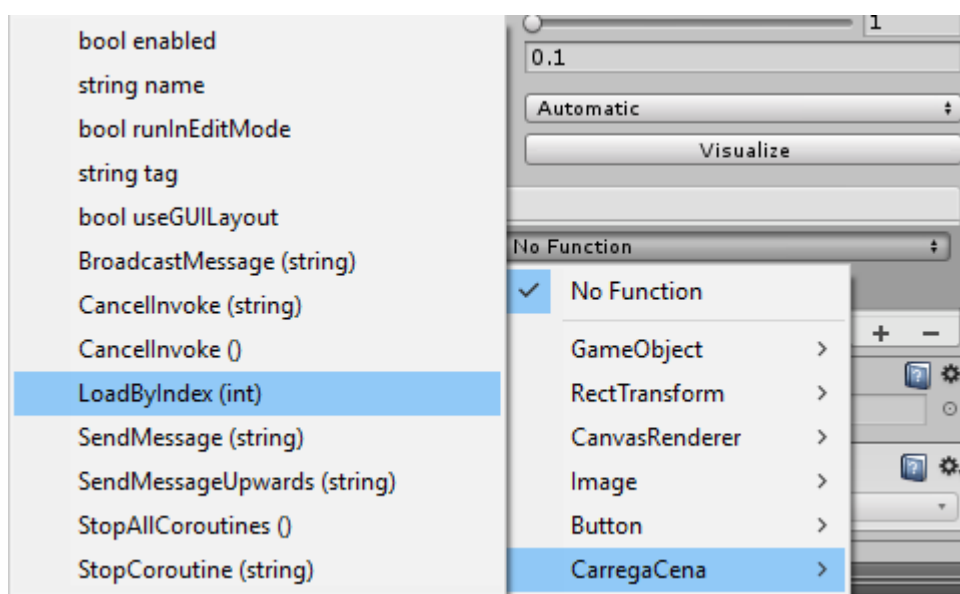


Figura A.97 – Seleção da opção “LoadByIndex (int)” para o evento ao clicar.

192. Repita o passo 189 para criar outro evento;

193. Repita o passo 29 para criar um *script* C# (dentro da pasta “Scripts”) e nomeie-o como “SalvarTempoEPontos”. Feito isto, abra o *script* recém-criado, apague o conteúdo padrão e copie o código do apêndice N para o *script* e salve. Volte para a tela do Unity 3D;
194. Insira o *script* recém-criado como mostra o passo 190 no objeto de jogo “voltarMenuPrincipal” e complete as variáveis do *script* recém-criado com os objetos de jogo “infoDesempenho” e “infoTempo”, como mostrado na Figura A.98;



Figura A.98 – Componente “Salvar Tempo E Pontos (Script)” de um objeto de jogo.

195. Repita o passo 190 e adicione o novo componente ao segundo evento;
196. Clique em “No Function” e, então, selecione “SalvarTempoEPontos”. Assim, selecione a opção “SaveButtonOnClick ()”, como mostra a Figura A.99;

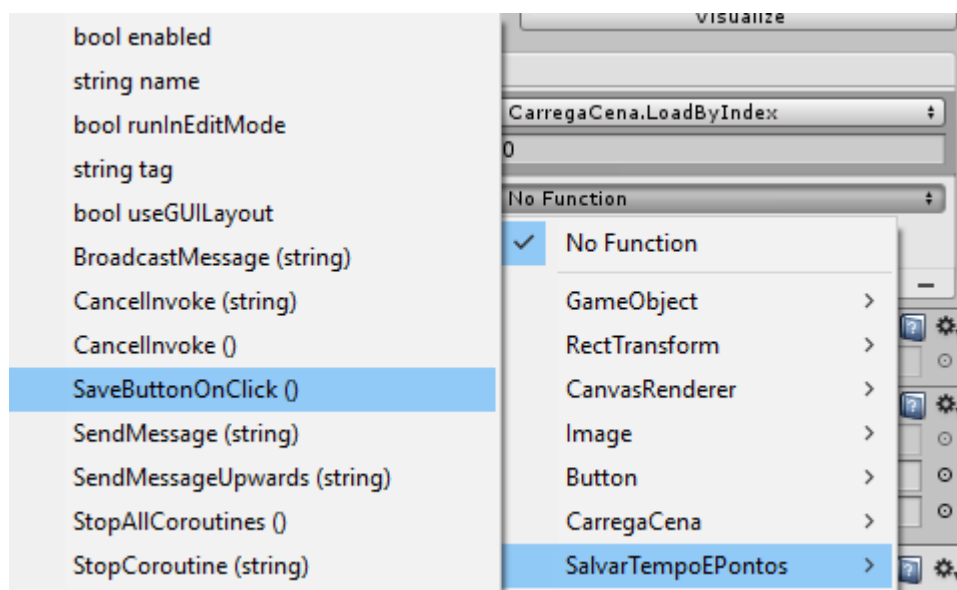


Figura A.99 – Seleção da opção “SaveButtonOnClick ()” para evento ao clicar.

Assim, finaliza-se o tutorial do jogo desenvolvido neste trabalho. Alguns pontos foram simplificados ou não detalhados devido ao fato de que são apenas questões estéticas que foram implementadas apenas a título de curiosidade e conhecimento melhor do software.

A verificação de execução do jogo, deste que o mesmo esteja com nenhum procedimento em aberto (por exemplo, não há componentes a serem completados com variáveis) é feita apertando o botão de “Play”. Para pausar a execução ou encerrar, utiliza-se os botões específicos para estas ações. Todos estes botões de controle de execução estão destacados na Figura A.100.

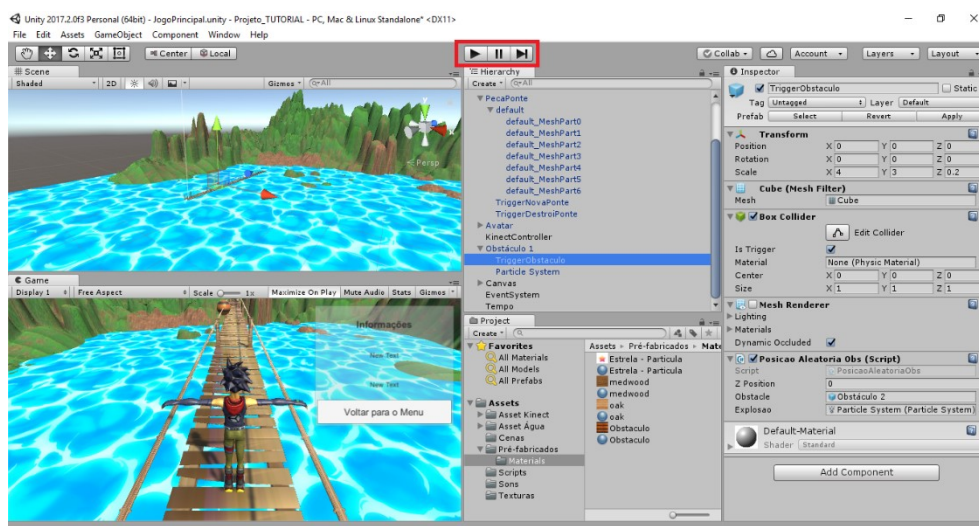


Figura A.100 – Destaque aos controles de execução do jogo.

É possível criar um executável do jogo desenvolvido no Unity 3D. Para isto, vá ao menu File e então clique na opção “Build Settings...” (Ctrl+Shift+B) e, na janela que irá aparecer, como mostra a Figura A.101, selecione a plataforma desejada. Neste caso, desenvolveu-se um jogo para Windows com uma arquitetura tanto de 32 bits quanto de 64 bits. Não é necessário nenhum método de compressão, pois o jogo não é extremamente complexo. Então, deve-se clicar em “Build” e aguardar a criação do arquivo executável.

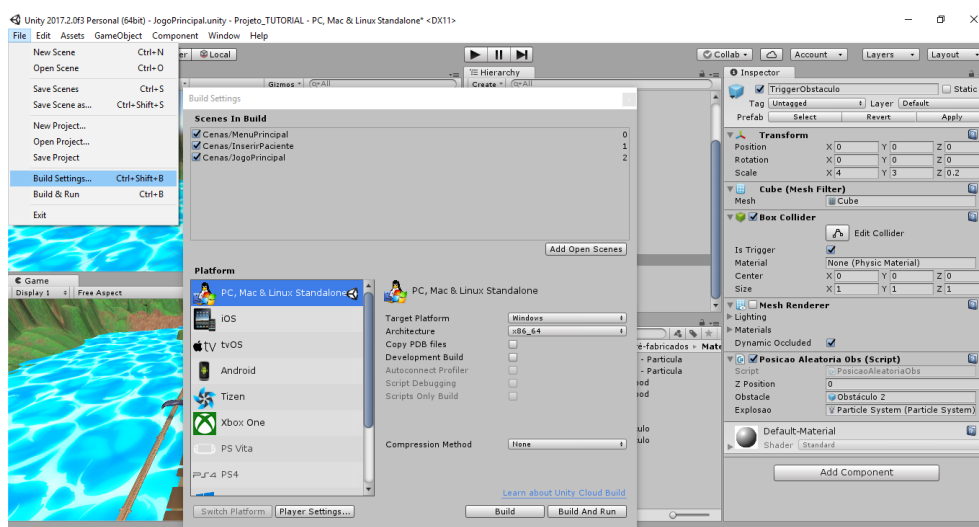


Figura A.101 – Janela de configurações de criação de um arquivo executável para o jogo desenvolvido.

B – TUTORIAL PARA OBTENÇÃO DO ASSET “Kinect v2 Examples MS-SDK”

Este *asset* é disponibilizado na rede pelo seu desenvolvedor, Rumen Filkov, pesquisador da *Vorarlberg University of Applied Sciences*, na Áustria. Para obtê-lo deve-se encaminhar um e-mail para “rumen.filkov@gmail.com” solicitando acesso para download do “Kinect v2 Examples MS-SDK”, partindo de um e-mail institucional, uma vez que ele só é disponibilizado gratuitamente para fins acadêmicos.

Este *asset* não é disponibilizado pelo autor deste projeto uma vez que o criador do *asset* não permite o compartilhamento do mesmo sem consentimento explícito.

C – TUTORIAL PARA OBTENÇÃO DO ASSET “Water Flow FREE”

Acesse a *Asset Store* da *engine* Unity 3D e faça *login* em sua conta. No mecanismo de busca da própria loja, procure pelo termo Water Flow FREE (Disponível em: <https://assetstore.unity.com/packages/vfx/shaders/water-flow-free-26434> – Acesso em 23/01/2018). Clique em “Open in Unity”, como mostra a Figura C.1.

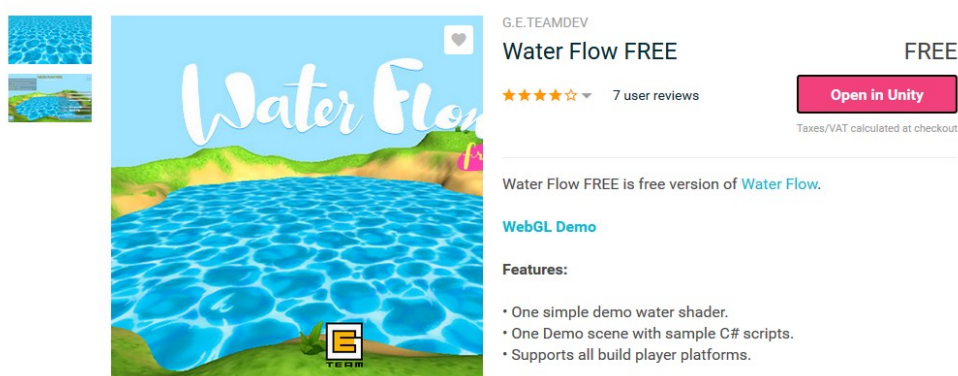


Figura C.1 – Visualização de parte da tela de aquisição do asset “Water Flow FREE”.

Esta ação, abrirá a *Asset Store* dentro do Unity 3D e, então, será possível fazer o download, como mostra a Figura C.2, e clique na opção.

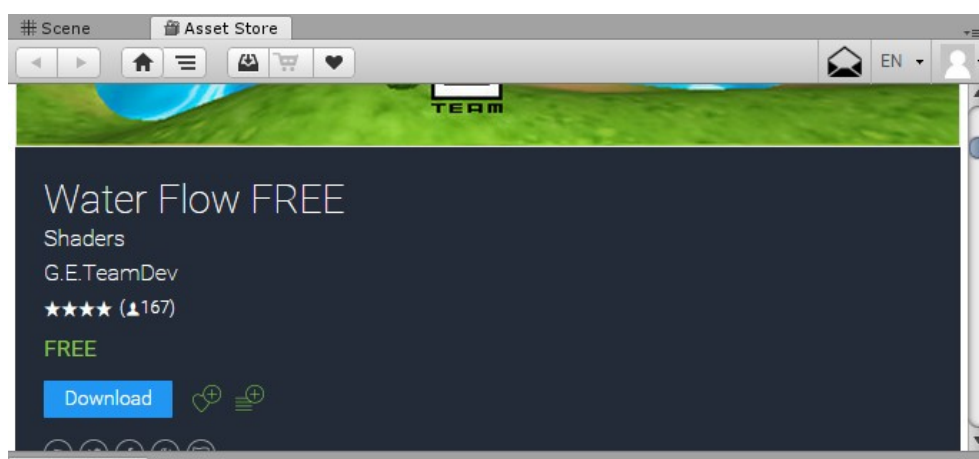


Figura C.2 – Caixa de download dentro do software Unity 3D para download do asset.

Neste caso, somente serão necessários os elementos marcados na Figura C.3.

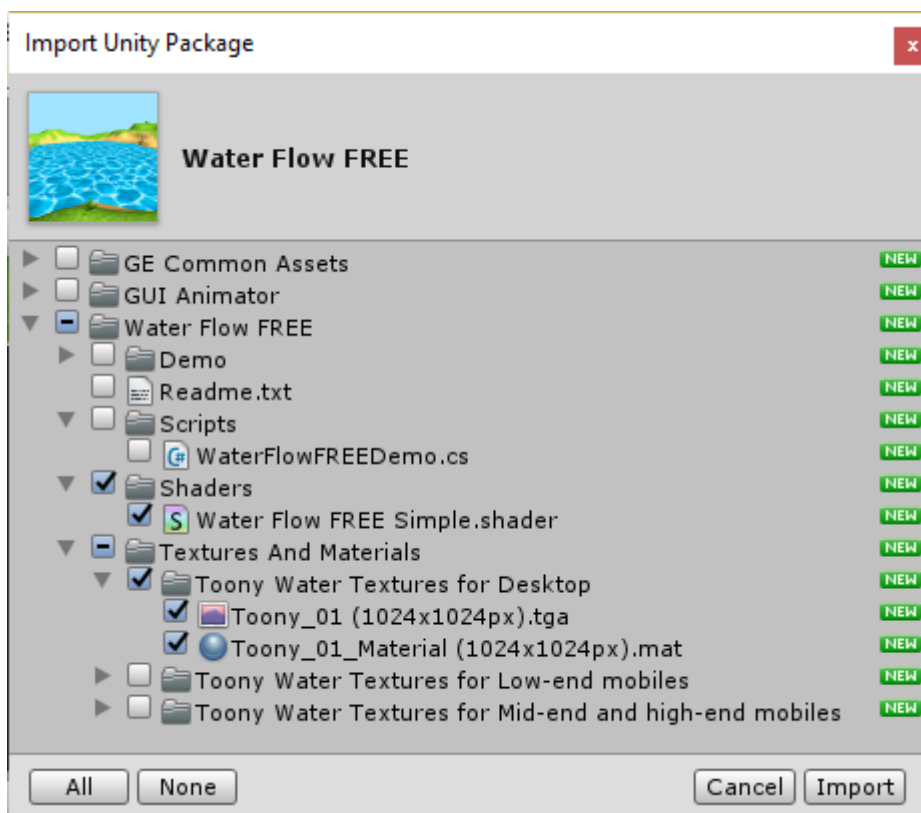


Figura C.3 – Arquivos que serão necessários para a utilização no desenvolvimento do jogo.

Feita a importação do *asset*, garanta que todo o conteúdo importado esteja dentro da pasta criada com o nome de “Asset Água”.

D - SCRIPT

CarregaCena.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine.SceneManagement;
using UnityEngine;

public class CarregaCena : MonoBehaviour
{
    public void LoadByIndex (int sceneIndex)
    {
        SceneManager.LoadScene (sceneIndex);
    }
}
```

E - SCRIPT

CriaPonte.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class CriaPonte : MonoBehaviour
{
    public Transform pontePrefab;

    // Quando entrar dentro do Trigger, executa esta função
    void OnTriggerEnter (Collider other)
    {
        Instantiate (pontePrefab, new Vector3 (750.25f, -850f ,
27.4f), pontePrefab.rotation);
    }
}
```

F - SCRIPT

DestroiObstaculo.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class DestroiObstaculo : MonoBehaviour
{
    public float tempoCorrente;
    public float tempoDeDestruicaoDoObstaculo;
    public Renderer rend;

    // Use this for initialization
    void Start ()
    {
        rend = GetComponent <Renderer> ();
        rend.enabled = false;
        tempoDeDestruicaoDoObstaculo = 10.0f;
    }

    // Update is called once per frame
    void Update ()
    {
        tempoCorrente += Time.deltaTime;

        if (tempoCorrente > tempoDeDestruicaoDoObstaculo)
        {
            Destroy (gameObject);
        }
    }
}
```

G - SCRIPT

DestroiPonte.cs

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class DestroiPonte : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }

    void OnTriggerEnter (Collider other)
    {
        Destroy (transform.parent.gameObject);
    }

}
```

H - SCRIPT

LevelLoader.cs

```

using System.Collections;
using UnityEngine;
using UnityEngine.SceneManagement;
using UnityEngine.UI;

public class LevelLoader : MonoBehaviour
{
    public Slider slider;
    public GameObject loadingMenu;
    public GameObject mainMenu;
    public Text textProgress;

    public void LoadLevel (int sceneIndex)
    {
        StartCoroutine (AsynchronousLoad (sceneIndex));
    }

    IEnumerator AsynchronousLoad (int sceneIndex)
    {
        yield return null;

        AsyncOperation ao = SceneManager.LoadSceneAsync (sceneIndex);

        mainMenu.SetActive (false);
        loadingMenu.SetActive (true);

        while (! ao.isDone)
        {
            float progress = Mathf.Clamp01 (ao.progress / 0.9f);

            slider.value = progress;
            textProgress.text = progress * 100f + "%";

            yield return null;
        }
    }
}

```

I - SCRIPT

```

Mare.cs
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Mare : MonoBehaviour
{
    IEnumerator Start ()
    {
        Vector3 maxMare = transform.position + new Vector3 (0,
3.0f, 0);
        var minMare = transform.position;

        while (true) {
            yield return StartCoroutine (movimentoMare
(transform, minMare, maxMare, 5.0f));
            yield return StartCoroutine (movimentoMare
(transform, maxMare, minMare, 5.0f));
        }
    }

    IEnumerator movimentoMare ( Transform thisTransform, Vector3
inicioPos, Vector3 fimPos, float tempo)
    {
        var i = 0.0f;
        var rate = 1f/tempo;
        while (i < 1.0f)
        {
            i += Time.deltaTime * rate;
            thisTransform.position = Vector3.Lerp (inicioPos,
fimPos, i);
            yield return null;
        }
    }
}

```

J - SCRIPT

```

PosicaoAleatoriaObs.cs
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class PosicaoAleatoriaObs : MonoBehaviour {

    private Vector3 [] positionObstacles = new Vector3 [2];
    public float zPosition;
    string auxPosition;

    public GameObject obstacle;
    public ParticleSystem explosao;

    int randomInt;

    void Start()
    {
        if (PlayerPrefs.HasKey ("DistanciaObs"))
        {
            auxPosition = PlayerPrefs.GetString ("DistanciaObs
");
            if (auxPosition == "")
            {
                zPosition = 20.0f;
            } else
            {
                zPosition = float.Parse (auxPosition);
            }

            if (zPosition < 10.0f || zPosition > 100.0f)
            {
                zPosition = 20.0f;
            }
        } else
        {
            Debug.Log ("Valor padrão");
            zPosition = 20.0f;
        }

        positionObstacles [0] = new Vector3(-0.8f, 1.3f,
zPosition);
        positionObstacles [1] = new Vector3(0.8f, 1.3f,
zPosition);
    }

    void OnTriggerEnter (Collider other)
    {
        randomInt = Random.Range (0, 2);

        obstacle.transform.position = positionObstacles
[randomInt];
    }
}

```

```
        Instantiate (obstacle,  obstacle.transform.position,  
obstacle.transform.rotation);  
  
transform.parent.gameObject.AddComponent<DestroiObstaculo> ();  
        explosao.Play ();  
  
    }  
  
}
```

L - SCRIPT

```
Sair.cs
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class Sair : MonoBehaviour
{
    public void sairJogo()
    {
        #if UNITY_EDITOR
            UnityEditor.EditorApplication.isPlaying = false;
        #else
            Application.Quit ();
        #endif
    }
}
```

M - SCRIPT

```

SalvarArquivo.cs
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.IO;
using UnityEngine.UI;

public class SalvarArquivo : MonoBehaviour {

    string nomeArquivo;
    public Text nome;
    public Text genero;
    public Text idade;
    public Text inicio;
    public Text terapiaN;
    public Text hoje;
    public Text distancia;
    public Text caminhoArquivo;

    public void SaveUser (string nameUser)
    {
        nomeArquivo = caminhoArquivo.text + "\\\" + nameUser + " -
Sessão #" + terapiaN.text + ".txt";
        PlayerPrefs.SetString ("fileName", nomeArquivo);

        StreamWriter mySW = new StreamWriter (nomeArquivo, true);

        // tópicos básicos do arquivo
        mySW.WriteLine ("Nome: " + nome.text);
        mySW.WriteLine ("Gênero: " + genero.text);
        mySW.WriteLine ("Idade: " + idade.text);
        mySW.WriteLine ("Data de início do tratamento: " +
inicio.text);
        mySW.WriteLine ("Terapia de número: " + terapiaN.text);
        mySW.WriteLine ("Data de hoje: " + hoje.text);
        mySW.WriteLine ("Distância entre obstáculos (entre 10 e
100): " + distancia.text);

        PlayerPrefs.SetString ("DistanciaObs", distancia.text);

        mySW.Flush ();
        mySW.Close ();

        Debug.Log ("SaveUser()");
    }

    public void SaveButtonOnClick()
    {
        if (nome.text == "") {
            Debug.Log ("Entre com um nome para o paciente.");
            return;
        } else
        {
            Debug.Log ("SaveButtonOnClick()");
            GetComponent<SalvarArquivo> ().SaveUser (nome.text);
        }
    }
}

```

```
}  
}
```

N - SCRIPT

```
SalvarTempoEPontos.cs
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.IO;
using UnityEngine.UI;

public class SalvarTempoEPontos : MonoBehaviour {

    string fileName;
    public Text points;
    public Text executionTime;

    void SaveTP () {

        fileName = PlayerPrefs.GetString ("fileName");

        StreamWriter mySW = new StreamWriter (fileName, true);

        mySW.WriteLine ("Tempo de duração da sessão atual: " +
executionTime.text);
        mySW.WriteLine ("Pontuação da sessão atual: " +
points.text);

        mySW.Flush ();
        mySW.Close ();

    }

    public void SaveButtonOnClick()
    {
        GetComponent<SalvarTempoEPontos> ().SaveTP ();
    }

}
```

O - SCRIPT

```

TempoExecucao.cs
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
using System.IO;

public class TempoExecucao : MonoBehaviour {

    float tempoSeg = 00;
    int tempoMin = 00, tempoHor = 00;
    string auxiliarSeg, auxiliarMin, auxiliarHor;
    public Text execution;

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

        tempoSeg = Time.timeSinceLevelLoad;

        tempoHor = (tempoMin / 60);
        tempoMin = ((int)tempoSeg / 60);
        tempoSeg = tempoSeg % 60;

        auxiliarSeg = tempoSeg.ToString ("##.##");
        auxiliarMin = tempoMin.ToString ("##");
        auxiliarHor = tempoHor.ToString ("##");

        if (tempoHor == 0 && tempoMin == 0)
        {
            execution.text = "00hrs00min" + auxiliarSeg + "s";
        } else if (tempoHor == 0 && tempoMin != 0)
        {
            execution.text = "00hrs" + auxiliarMin + "min" +
auxiliarSeg + "s";
        } else
        {
            execution.text = auxiliarHor + "hrs" + auxiliarMin
+ "min" + auxiliarSeg + "s";
        }
    }
}

```

P - SCRIPT

VariacaoVelocidadeAvatar.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;

public class VariacaoVelocidadeAvatar : MonoBehaviour
{
    public float velocidadePassada;
    public Text desempenhoAvatar;
    public float fatorDeCorrecao;

    public float velocidadePassadaAnt = 0;
    public int soma = 0;
    public int limite = 0;

    IEnumerator Start ()
    {
        fatorDeCorrecao = 1.6f;
        while (true) {
            yield return StartCoroutine (delayPassada (0.05f));
        }
    }

    IEnumerator delayPassada (float tempo)
    {
        Vector3 inicial = transform.position;
        var i = 0.0f;
        var rate = 1f/tempo;
        while (i < 1.0f)
        {
            i += Time.deltaTime * rate;
            yield return null;
        }
        Vector3 final = transform.position;
        velocidadePassada = (Mathf.Abs((inicial.magnitude -
final.magnitude) / 0.05f))*fatorDeCorrecao;

        if (limite == 25)
        {
            if (velocidadePassada < 0.01) {
                soma = soma;
            } else {
                soma = soma + (Mathf.CeilToInt
(velocidadePassada) + Mathf.CeilToInt (velocidadePassadaAnt));
            }
            desempenhoAvatar.text = soma + " pontos!";
            velocidadePassadaAnt = velocidadePassada;

            limite = 0;
        }
        limite = limite + 1;
    }
}

```

Q - SCRIPT

VelocidadePonte.cs

```

using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class VelocidadePonte : MonoBehaviour
{
    public Vector3 variableVelocity;
    public float speedRunner;
    private GameObject dedoEsquerdo;
    private GameObject dedoDireito;

    // Use this for initialization
    void Start ()
    {
        dedoEsquerdo = GameObject.Find
("Avatar/U/joint_Char/joint_Pelvis/joint_HipMaster/joint_HipLT/joint_Knee
LT/joint_FootLT/joint_ToeLT");
        dedoDireito = GameObject.Find
("Avatar/U/joint_Char/joint_Pelvis/joint_HipMaster/joint_HipRT/joint_Knee
RT/joint_FootRT/joint_ToeRT");
    }

    // Update is called once per frame
    void Update ()
    {
        speedRunner =
(dedoDireito.GetComponent<VariacaoVelocidadeAvatar> ().velocidadePassada +
dedoEsquerdo.GetComponent<VariacaoVelocidadeAvatar>
().velocidadePassada)/2;
        variableVelocity = new Vector3 (0, 0, speedRunner);
        transform.position -= variableVelocity * Time.deltaTime;
    }
}

```

R – PROCEDIMENTO DE CONVERSÃO ENTRE ARQUIVOS “.stl” E ARQUIVOS “.obj”

Os arquivos no formato “.stl” podem ser obtidos no software SolidWorks utilizando a opção “Salvar como”, como mostrado na Figura R.1.

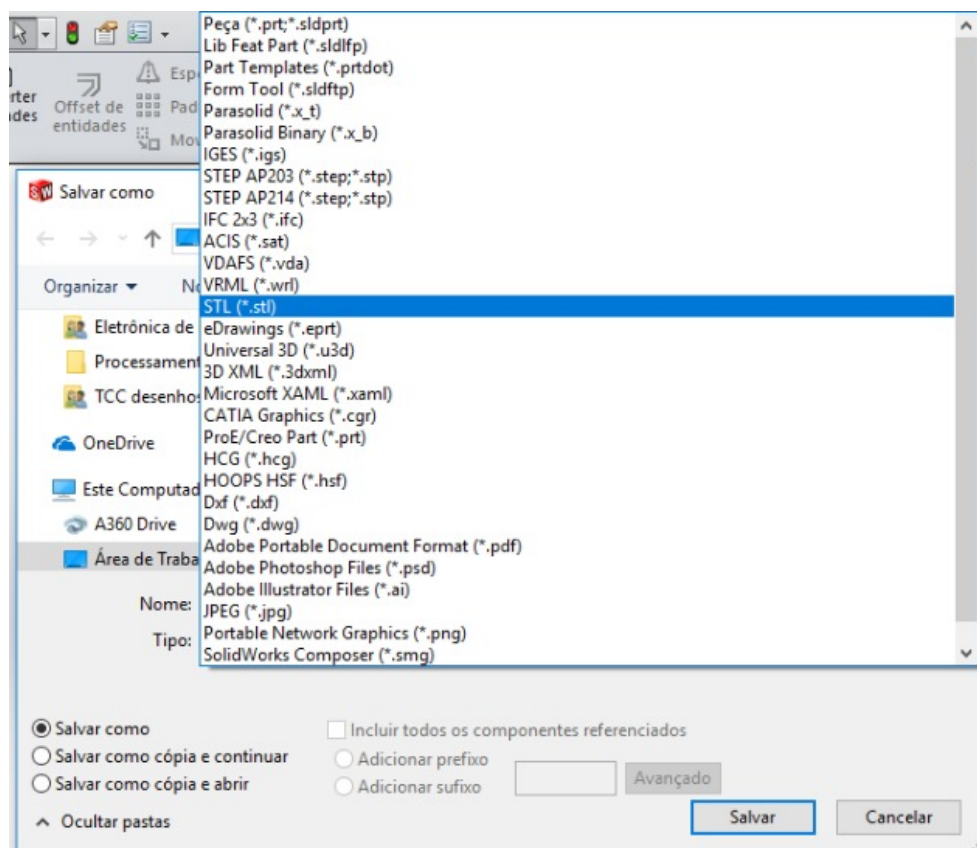


Figura R.1 – Seleção do formato “.stl” para salvar a peça construída desta maneira.

E então, utilizando o software 3D Builder, software gratuito distribuído juntamente com o Windows 10, é possível importar o modelo em formato “.stl”. Ao importar o modelo, é necessário escolher a unidade de medida utilizada no desenvolvimento da peça no software de origem (no caso, SolidWorks).

Para adicionar as texturas necessárias deve simplesmente selecionar o modelo, ir até o menu “Tinta” e então escolher a opção “Textura”. Para aplicar a textura, deve-se escolher a mesma no menu destinado a elas e, então, clicar em cada uma das partes para adicioná-la a mesma ao modelo. Neste projeto, foram adicionadas as texturas “oak” e “medwood”, como mostrado na Figura R.2.

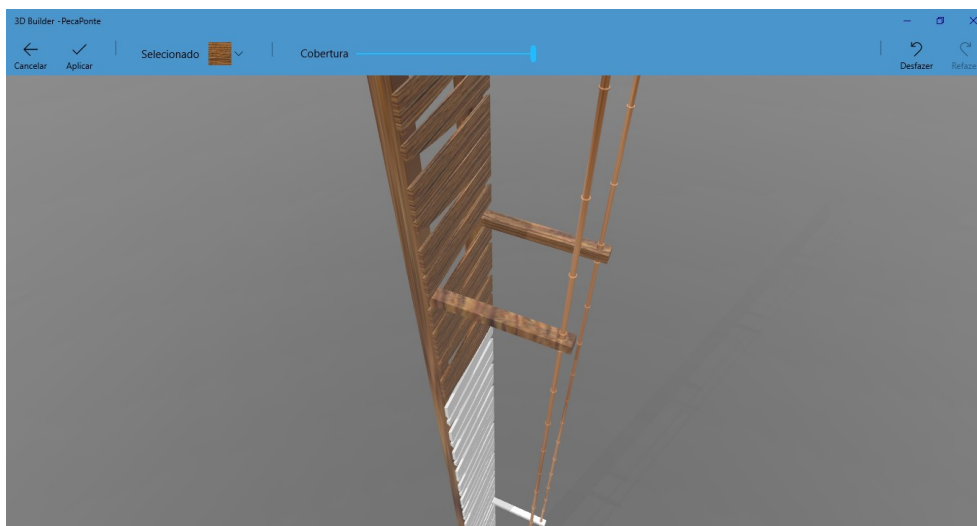


Figura R.2 – Seleção das texturas para os componentes do modelo 3D importado para o 3D Builder.

Após, deve-se aplicar as texturas no botão “Aplicar”. Ir na opção “Salvar como” e então escolher o formato “.obj”, como mostra a Figura R.3.

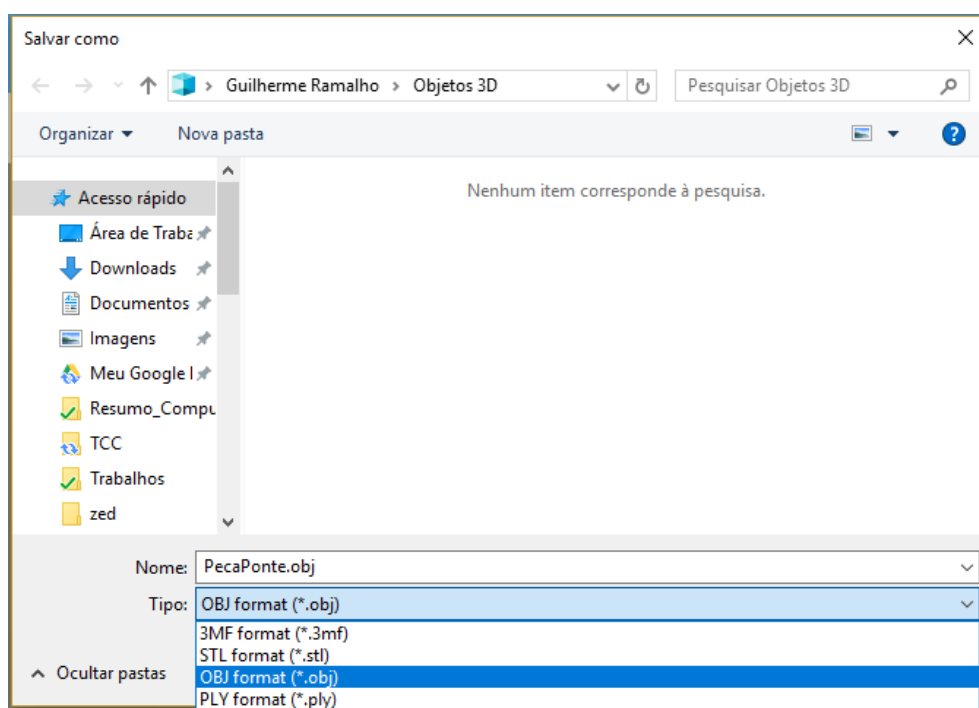


Figura R.3 - Seleção do formato “.obj” para salvar o modelo importado desta maneira.

S – FORMULÁRIO PARA APLICAÇÃO DO GAME QUESTIONNAIRE EXPERIENCE

O formulário foi desenvolvido na plataforma Forms® da empresa Google. Porém, neste anexo, o mesmo está transcrito integralmente a seguir.

Testes de experiência - Game Experience Questionnaire

Este formulário é parte integrante do projeto de fim de curso "DESENVOLVIMENTO DE UM JOGO SÉRIO PARA A REABILITAÇÃO DA MARCHA HUMANA" e conta como objetivo o desenvolvimento de uma base de dados para medir a experiência humana dentro de mídias digitais.

As respostas do jogo seguem o grau de concordância da escala de Likert:

- (1) De modo algum;
- (2) Levemente;
- (3) Moderadamente;
- (4) Bastante;
- (5) Extremamente.

Serão analisados sete aspectos de imersão:

- 1 - Desafio;
- 2 - Competência;
- 3 - Fluidez;
- 4 - Efeito Negativo;
- 5 - Efeito Positivo;
- 6 - Imersão Sensorial e Significativa;
- 7 - Tensão

Eu me senti desafiado:

()1 ()2 ()3 ()4 ()5

Eu me senti habilidoso:

()1 ()2 ()3 ()4 ()5

Eu me senti imerso:

()1 ()2 ()3 ()4 ()5

Eu o achei cansativo:

()1 ()2 ()3 ()4 ()5

Eu o achei divertido:

()1 ()2 ()3 ()4 ()5

O jogo era esteticamente agradável:

()1 ()2 ()3 ()4 ()5

Eu me senti frustrado:

()1 ()2 ()3 ()4 ()5