UNIVERSIDADE FEDERAL DE UBERLÂNDIA

RODRIGO ALVES PRADO

DESENVOLVIMENTO DE JOGOS SÉRIOS PARA REABILITAÇÃO DA COLUNA HUMANA

Uberlândia

2021

RODRIGO ALVES PRADO

DESENVOLVIMENTO DE JOGOS SÉRIOS PARA REABILITAÇÃO DA COLUNA HUMANA

Projeto de Fim de Curso apresentado ao curso de Graduação em Engenharia Mecatrônica da Universidade Federal de Uberlândia, como um dos requisitos para obtenção do título de ENGENHERO MECATRÔNICO.

Área de concentração: Robótica.

Orientador: Prof. Dr. Rogério Sales Gonçalves.

UBERLÂNDIA

2021

Dedico esta conquista aos meus pais, Idelma e Marcelo, meus irmãos, Marcelo e Rafael, à minha namorada Bruna, aos meus avós José, Maria, Rogério e em especial à minha avó Adair que deixou muita saudade e um grande exemplo de vida e humildade. Sempre me deram apoio, carinho e força para seguir em frente e nunca desistir dos meus sonhos. Minha gratidão é eterna a vocês. Gostaria, também, de dedicar aos pacientes que possam vir a se beneficiar deste projeto.

AGRADECIMENTOS

Agradeço primeiramente a Deus por me guiar na caminhada e sempre me manter forte para continuar minha jornada sem temer o que viria à frente.

Toda minha família, em especial meus pais Idelma Ely Alves Prado e Marcelo Magno Prado pelo carinho, amor, apoio e incentivo em todas as etapas dessa jornada. Agradeço aos meus irmão Marcelo Alves Prado e Rafael Alves Prado por estarem ao meu lado e apostos sempre que precisei de ajuda. Agradeço aos meus avós, em especial minha vó Adair Ângela alves que foi chamada para alegrar os céus, assim como sempre nos alegrou aqui na Terra, obrigado por tudo vó.

A minha namorada Bruna Rafaela Santos Guimarães pelo carinho, confiança e suporte nesta caminhada, sempre me incentivando a continuar, apoiando nos momentos difíceis e compartilhando os momentos felizes, obrigado por toda paciência e por todo amor.

A Universidade Federal de Uberlândia pelo ensino de qualidade, suporte durante o curso e pelas diversas oportunidades de expandir o conhecimento no curso.

Agradeço ao Professor Doutor Rogério Sales Gonçalves pelos ensinamentos, orientação e direcionamento para a realização deste trabalho, além da oportunidade de fazer algo que possa aliviar a dor e incentivar as pessoas durante seus tratamentos.

Agradeço aos meus amigos pelo incentivo e auxilio, e a todas as pessoas que me apoiaram de alguma maneira durante a realização deste trabalho.

Agradeço em especial ao meu irmão Marcelo Alves Prado e aos meus amigos Matheus Chaves D’ Carvalho e Brian Batista pelo apoio e pelos ensinamentos de programação e utilização da Unity, cada palavra de vocês me trouxe um passo mais perto de concluir este trabalho.

Agradeço aos meus companheiros de apartamento da República Mandelaje pelo apoio e por sempre estarem presentes nos momentos felizes e principalmente nos momentos tristes dizendo que tudo ia dar certo no final, e deu! Obrigado a todos.

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

“A persistência é o caminho do êxito.”

— Marcelo Magno Prado

ALVES PRADO, R**. Desenvolvimento de jogos sérios para a reabilitação da coluna humana**. 2021. 180p. Monografia de Conclusão de Curso (Bacharel) Universidade Federal de Uberlândia, Uberlândia, 2021.

RESUMO

O objetivo deste projeto é o desenvolvimento de jogos eletrônicos sérios que possam auxiliar profissionais da saúde na fisioterapia de pacientes com problemas de dores na coluna vertebral. As dores crônicas da coluna são a segunda condição de saúde mais identificada por médicos ou profissionais de saúde no Brasil. Apesar da grande indicação e dos benefícios da fisioterapia para esses casos, o longo tempo de tratamento e a pouca motivação gerada pelos métodos tradicionais são umas das principais causas da falha terapêutica. Os jogos a serem desenvolvidos, que serão controlados pelo Openpose e desenvolvidos no software Unity, deverão funcionar como uma ferramenta lúdica para motivar os pacientes durante a fisioterapia e reduzir os casos de abandono do tratamento.

Palavras Chave: Reabilitação da coluna vertebral. Jogos sérios. Dores crônicas. Fisioterapia.

ALVES PRADO, R**. Development of serious games for the rehabilitation of the human spine**. 2021. 180p. Course Conclusion Monograph (Bachelor) Federal University of Uberlândia, Uberlândia, 2021.

ABSTRACT

The objective of this project is the development of serious games that can help health professionals in the physiotherapy of patients with spinal pain problems. Chronic back pain is the second most identified health condition by physicians or health professionals in Brazil. Despite the great indication and benefits of physiotherapy for these cases, the long time of treatment and the little motivation generated by traditional methods are one of the main causes of therapeutic failure. The serious games to be developed, which will be controlled by Openpose and developed in the Unity software, should work as a playful tool to motivate patients during physiotherapy and reduce cases of treatment dropout.

Key Words: Rehabilitation of the spine. Serious games. Chronic pain. Physiotherapy.

LISTA DE FIGURAS

LISTA DE SÍMBOLOS E ABREVIAÇÕES

UFU – Universidade Federal de Uberlândia

OMS – Organização Mundial da Saúde

UI – *User Interface*

PNAD - Pesquisa Nacional por Amostra de Domicílios

IBGE - Instituto Brasileiro de Geografia e Estatística

RNM - Ressonância Nuclear Magnética

TC - Tomografia Computadorizada

SUMÁRIO

CAPÍTULO I

INTRODUÇÃO

A coluna vertebral constitui importante papel na postura, sustentação de peso, locomoção, proteção da medula espinhal e raízes nervosas. Apesar de sua importância para o ser humano, segundo a Pesquisa Nacional por Amostra de Domicílios (PNAD) do Instituto Brasileiro de Geografia e Estatística (IBGE), foram estimadas aproximadamente 21,6% de pessoas de 18 anos ou mais de idade (34,3 milhões) que referiram problema crônico de coluna no Brasil (IBGE, 2019).

Tais dados são preocupantes, uma vez que as repercussões das dores na coluna para o indivíduo são diversas, como por exemplo, alteração de humor, dificuldade para dormir, necessidade de auxílio para realização de tarefas do cotidiano e perda do status da capacidade de trabalhar (SANARE, 2014).

Esses problemas de coluna, também chamados popularmente por “dores nas costas”, englobam as cervicalgias, espondiloses, radiculopatias, transtornos dos discos intervertebrais, além de dores torácicas, lombares e ciáticas (OLIVEIRA et al., 2015).

Diante da quantidade e da diversidade de quadros, uma das maiores demandas do serviço de saúde são as dores da coluna (NASCIMENTO e COSTA, 2015).

Dentre os objetivos da fisioterapia em paciente com dores na coluna vertebral, destaca-se: a redução da dor, o aumento da amplitude de movimento, o alívio da tensão da musculatura afetada, a manutenção do equilíbrio e das funções da coluna (GARCIA et al., 2011.

Apesar da grande indicação e dos benefícios da fisioterapia, segundo Dias, Sampaio e Taddeo (2009), “o longo tempo necessário para o tratamento e a pouca motivação gerada pelos

métodos tradicionais são apontados como motivo de abandono do tratamento fisioterápico, caracterizando-se como uma das principais causas de falha terapêutica (DIAS et al., 2009).”

Dessa forma, o uso de jogos sérios utilizando o Openpose pode ser uma excelente opção de ferramenta lúdica para motivar os pacientes durante a fisioterapia.

O Openpose representa o primeiro sistema multi pessoa em tempo real a detectar conjuntamente os pontos chave do corpo humano, ele permite aos jogadores serem identificados pelo dispositivo e, assim, dar comandos nos jogos por meio de gestos, portanto sem a necessidade de um controle (joystick). É de autoria de Ginés Hidalgo, Zhe Cao, Tomas Simon, Shih-En Wei, Yaadhav Raaj, Hanbyul Joo e Yaser Sheikh e é mantido por Ginés Hidalgo e Yaadhav Raaj (HIDALGO et al., 2019). O OpenPose possui aplicação em diversas áreas, dentre as quais o campo da saúde/reabilitação pode se tornar amplamente utilizado.

Portanto, esse projeto de pesquisa tem como objetivo desenvolver jogos sérios controlados pelo OpenPose que possam auxiliar profissionais da saúde na fisioterapia de pacientes com problemas de dores na coluna vertebral.

Os jogos sérios devem solicitar que a pessoa a ser tratada realize os exercícios de uma sessão de fisioterapia convencional, porém de maneira bem mais lúdica. Tudo isso visando a melhora da qualidade de vida do paciente e redução de casos de abandono do tratamento de reabilitação da coluna vertebral.

1.1. Objetivos

1.1.1. Objetivo Geral

O objetivo geral deste projeto é desenvolver jogos sérios (*serious games*), utilizando o OpenPose e o software para desenvolvimento de jogos Unity 3D, com o intuito de auxiliar profissionais da saúde na fisioterapia de pacientes com problemas de dores na coluna vertebral. Os jogos sérios a serem desenvolvidos deverão funcionar como uma ferramenta lúdica para motivar os pacientes durante a fisioterapia e reduzir os casos de abandono do tratamento. Desta forma, deveram ser um recurso adicional e não substituto no processo de terapia destes pacientes.

1.1.2. Objetivos específicos

Os objetivos específicos para o projeto em questão são:

- Estudo dos movimentos da coluna e reabilitação desta;
- Estudo do OpenPose;
- Estudo do Unity 3D;
- Planejamento de jogos sérios;
- Estudo do desenvolvimento de jogos;
- Desenvolvimento dos cenários;
- Planejamento e adaptação do avatar;
- Desenvolvimento do controle e fluidez entre usuário e avatar;
- Desenvolvimento dos jogos;
- Desenvolvimento do menu principal;
- Desenvolvimento dos menus secundários;
- Desenvolvimento da tela de inserção dos dados do usuário;
- Determinação da tela de calibração;
- Desenvolvimento do sistema de pontuação dentro dos jogos;
- Desenvolvimento do sistema de marcação de tempo de execução dos jogos;

1.2. Justificativa

O mundo atual está se transformando cada vez mais rápido como consequência de um desenvolvimento tecnológico acelerado. Novas tecnologias proporcionam uma verdadeira revolução na sociedade, possibilitando, muitas vezes, uma melhora da qualidade de vida das pessoas.

Ademais, tais inovações tecnológicas revolucionaram também a forma como é feito o entretenimento. Evidência disso é o advento dos jogos eletrônicos. Conforme um levantamento realizado pela Newzoo, em 2017, o Brasil era o principal mercado de jogos da América Latina e décimo terceiro no ranking mundial (NEWZOO, 2017).

Por outro lado, as novas tecnologias emergentes não necessitam necessariamente se enquadrar em um desses dois campos: melhora da qualidade de vida humana ou entretenimento, mas podem, na verdade, abranger essas duas áreas de uma só vez. Essa é a proposta desse projeto, desenvolver jogos sérios cuja finalidade principal é auxiliar na fisioterapia da coluna vertebral, oferecendo alívio das "dores nas costas" e, por conseguinte, o bem-estar dos pacientes. No entanto, tudo isso com uma atmosfera de entretenimento, propiciada pelos elementos gráficos e sonoros dos jogos, permitindo, assim, uma experiência fisioterápica menos entediante e com menos casos de abandono do tratamento.

Além disso, os jogos que serão desenvolvidos se inserem no contexto dos *serious games* (jogos sérios). Esse termo é empregado para identificar jogos que extrapolam a finalidade de entretenimento, oferendo também outros tipos de experiências aos usuários, como direcionadas ao aprendizado ou ao treinamento (BARNEST,2009). Estes jogos podem ser aplicados em diversas áreas, como por exemplo, empresarial, militar, política e artes, todavia é exatamente na saúde que eles mais têm ganhado destaque (MACHADO, 2009).

CAPÍTULO II

FUNDAMENTAÇÃO TEÓRICA

Neste capítulo serão apresentadas as informações necessárias para o entendimento deste trabalho, tais como definições da coluna vertebral humana e problemas relacionados à mesma, procedimentos de reabilitação, jogos sérios, OpenPose e o software de desenvolvimento de jogos Unity 3D.

2.1. Coluna vertebral humana

Responsável por sustentar a posição bípede humana, a coluna vertebral constitui importante papel na postura, sustentação de peso, locomoção, proteção da medula espinhal e raízes nervosas. É formada por vértebras, medula espinhal e, também, por tecidos moles como músculos, ligamentos, cápsulas, tendões e discos, onde tais estruturas são responsáveis pela flexibilidade da coluna vertebral(MAGALHÃES, 2019).

2.1.1. Formação e composição da coluna vertebral

A extensão da coluna vertebral se dá desde a base do crânio até a extremidade caudal do tronco. É formada por vértebras empilhadas umas sobre as outras, onde as menores são as cervicais, as de tamanho mediano as toráxicas e, por fim, vértebras lombares são as maiores, estando localizadas na parte inferior da coluna.

O osso sacro é formado por vértebras sacrais, bem como o cóccix é formado pelas coccígeas. Os membros inferiores do ser humano se articulam na pelve, que é a base da coluna. A articulação da coluna se dá com o osso occipital do crânio na parte superior e, com o ilíaco na parte inferior(MAGALHÃES, 2019).

A coluna vertebral é constituída por 33 vértebras (MAGALHÃES, 2019) e, entre cada vértebra encontra-se um disco intervertebral. É formada por 5 divisões, são elas:

- Vértebras Cervicais: 7 vértebras;
- Vértebras Dorsais ou torácicas: 12 vértebras;
- Vértebras Lombares: 5 vértebras;
- Vértebras Sacrais: 5 vértebras fundidas;
- Vértebra Coccígea: 4 vértebras fundidas.

Na Figura 2.1 segue um desenho esquemático da divisão da coluna vertebral.

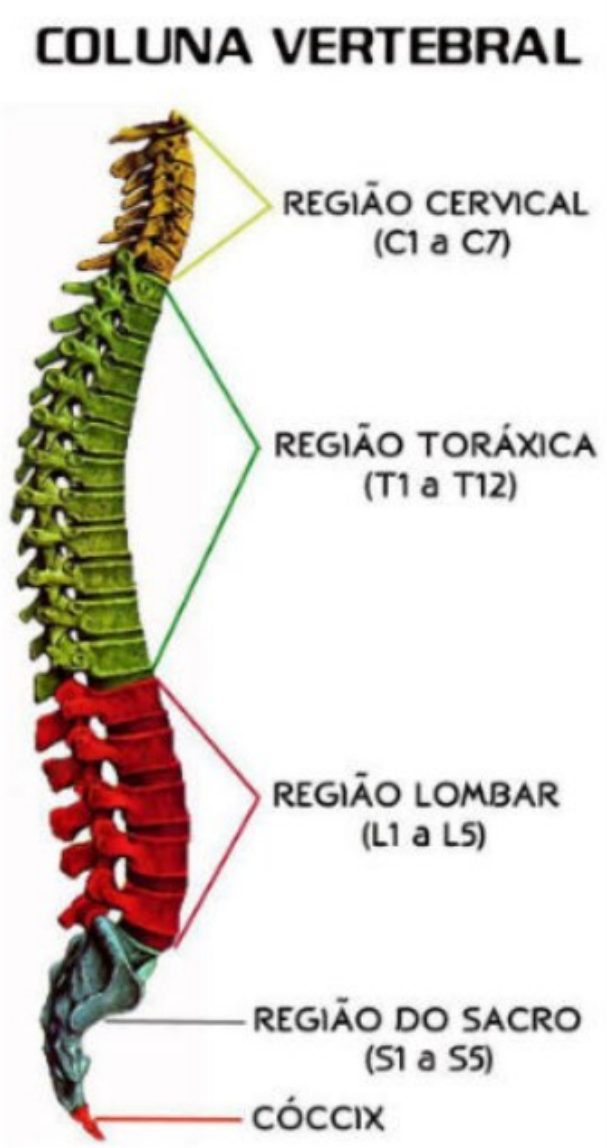

Figura 2.1 - Divisão da coluna vertebral humana (MAGALHÃES, 2019).

Para uma melhor visualização, é apresentado na Figura 2.2 a coluna vertebral em várias vistas com a localização e nomes das devidas regiões.

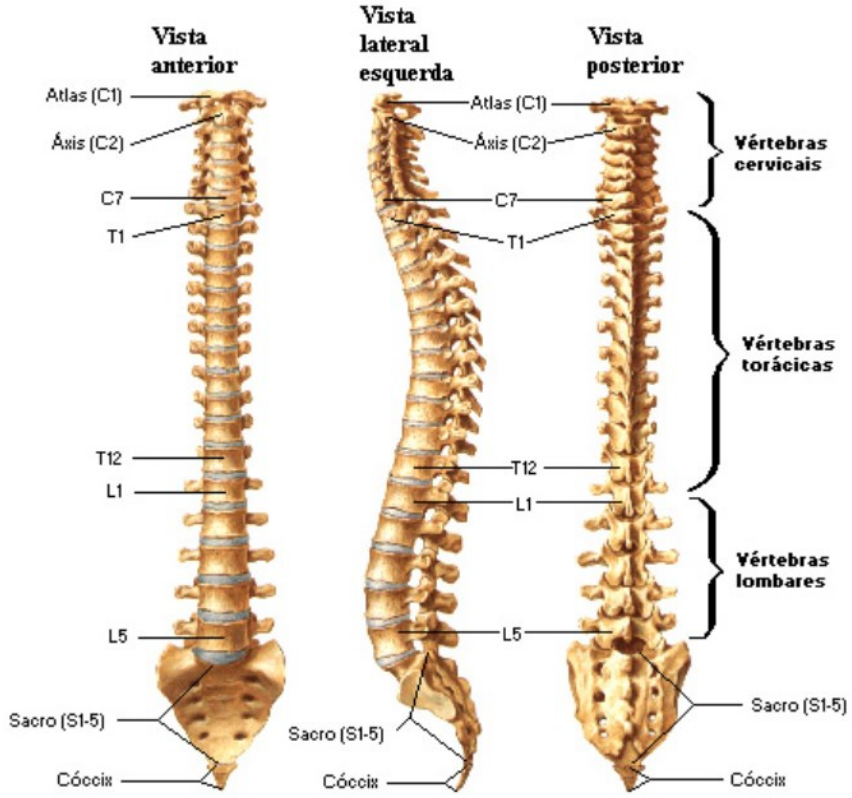

Figura 2.2 - Vistas da coluna vertebral humana (NETTER, 2000).

Os discos intervertebrais podem ser visualizados em detalhe na Figura 2.3:

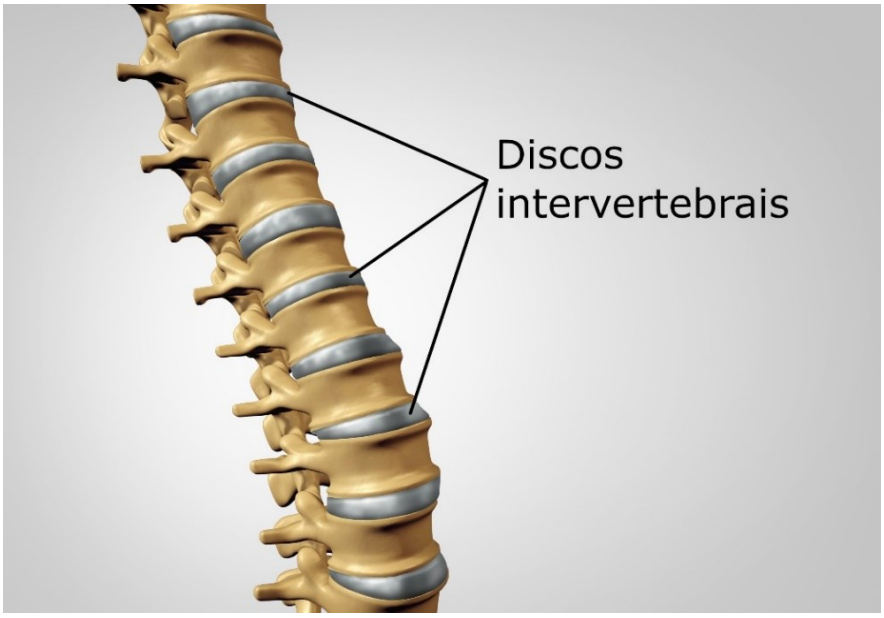

Figura 2.3 – Discos intervertebrais.
Fonte: https://www.infoescola.com/sistema-esqueletico/disco-intervertebral (Acesso em: 12/09/2021).

Com exceção da 1ª e 2ª vértebras cervicais, atlas (C1) e áxis (C2), respectivamente, todas as vértebras possuem 7 elementos básicos (MAGALHÃES, 2019):

- Corpo;
- Processo Espinhoso;
- Processo Transverso;
- Processos Articulares;
- Lâminas;
- Pedículos;
- Forame Vertebral.

O desenho esquemático básico da vertebra é apresentado na Figura 2.4.

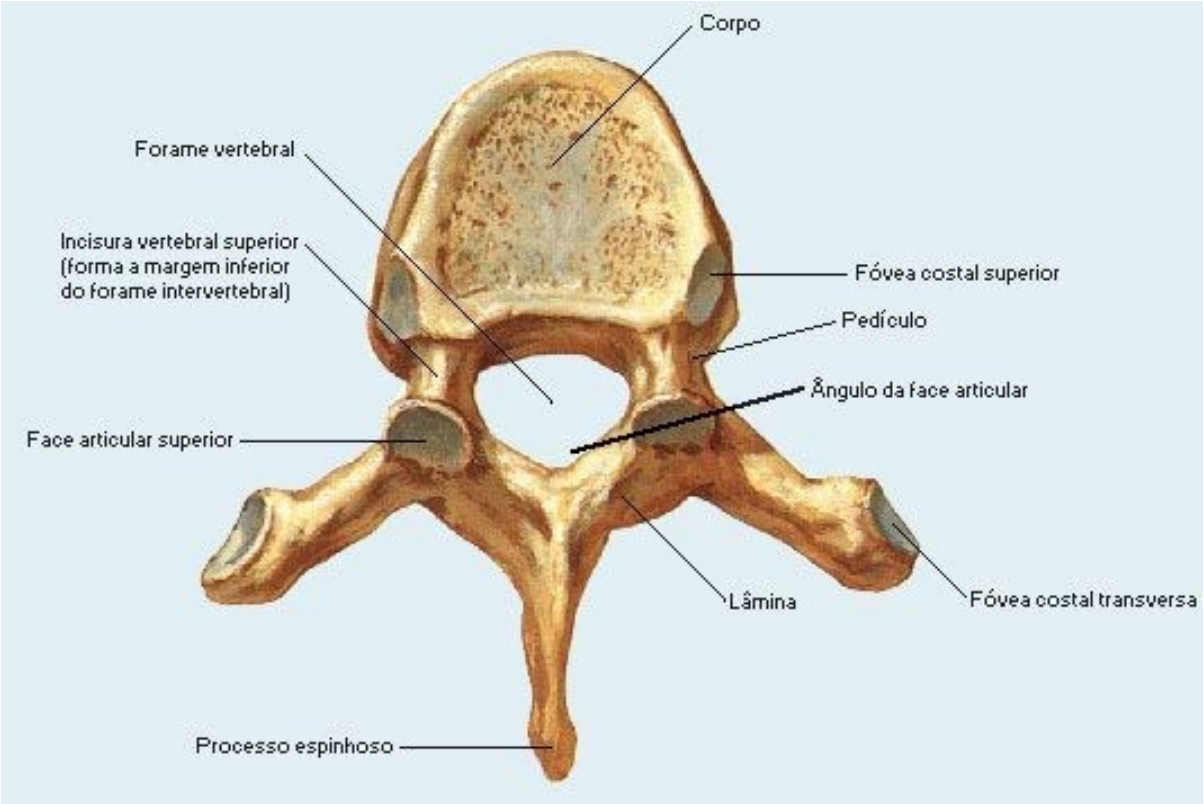

Figura 2.4 – Elementos que constituem as vértebras (MAGALHÃES, 2019).

A coluna vertebral do adulto apresenta quatro curvaturas sagitais: cervical, torácica, lombar e sacral. As curvaturas torácica e sacral, convexas posteriormente, são denominadas primárias porque apresentam a mesma direção da coluna vertebral fetal e decorrem da diferença de altura entre as partes anteriores e posteriores dos corpos vertebrais. As curvaturas cervical e lombar, côncavas posteriormente, formam-se após o nascimento e decorrem da

diferença de espessura entre as partes anteriores e posteriores dos discos intervertebrais (NATOUR et al., 2004).

1. Cervical: constitui o esqueleto axial do pescoço e suporte da cabeça.

2. Torácica: suporta a cavidade torácica.

3. Lombar: suporta a cavidade abdominal e permite mobilidade entre a parte torácica do tronco e a pelve.

4. Sacral: une a coluna vertebral à cintura pélvica.

A Figura 2.5 ilustra o arranjo anatômico geral da coluna vertebral.

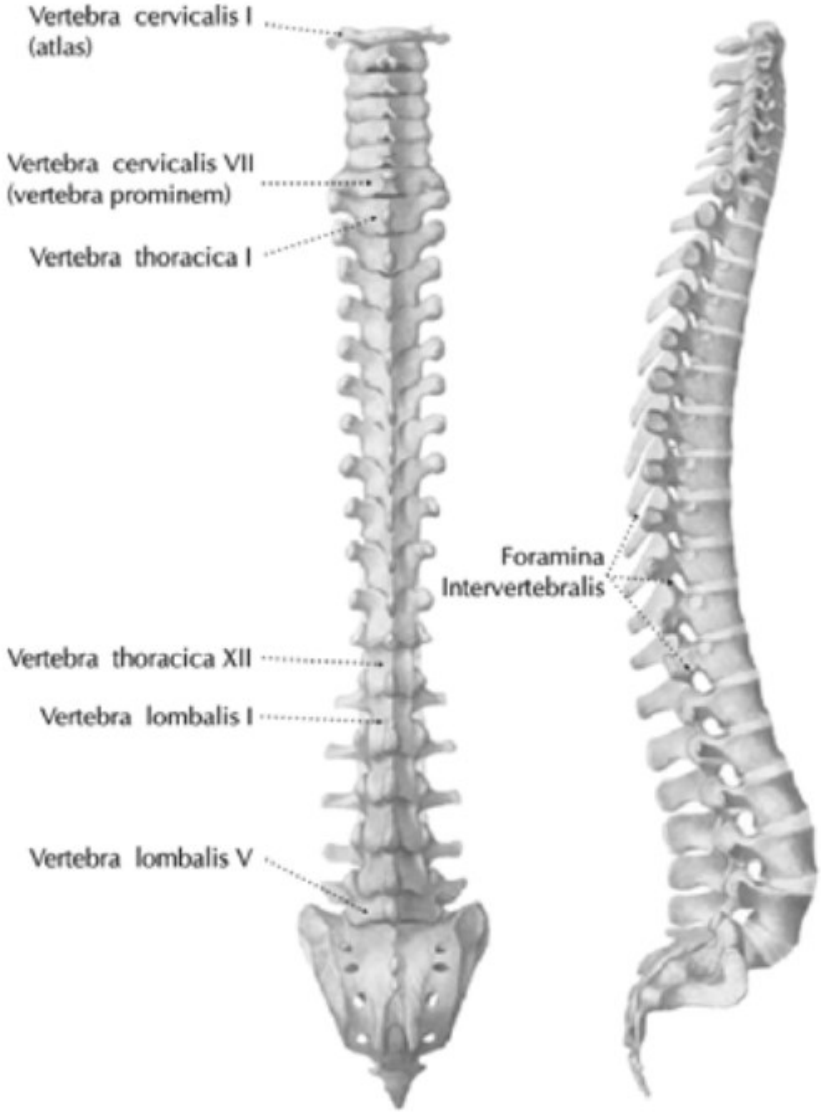

Figura 2.5 - Arranjo anatômico geral da coluna vertebral (NATOUR et al., 2004).
Fonte: https://bvsms.saude.gov.br/bvs/publicacoes/ColunaVertebral.pdf. Acesso em 11 de set. de 2021.

2.1.2. Movimentos da coluna vertebral

Somente movimentos limitados são possíveis entre vértebras adjacentes, mas a soma desses movimentos confere considerável amplitude de mobilidade na coluna vertebral como um todo. Movimentos de flexão, extensão, lateralização, rotação e circundação são todos possíveis, sendo essas ações de maior amplitude nos segmentos cervical e lombar que no torácico. Isso ocorre porque os discos intervertebrais cervicais e lombares apresentam maior espessura, não sofrem o efeito de contenção da caixa torácica, seus processos espinhosos são mais curtos e seus processos articulares apresentam forma e arranjo espacial diferente dos torácicos. A flexão é o mais pronunciado movimento da coluna vertebral (NATOUR et al., 2004).

O livro *"Coluna Vertebral"*, de Jamil Natour e colaboradores (2004), traz, em um de seus capítulos, um estudo sobre os movimentos da coluna vertebral, são eles:

1. Plano sagital
   – Flexão
   – Extensão
2. Plano coronal
   – Lateralização direita
   – Lateralização esquerda
3. Plano longitudinal
   – Rotação ou circundação

Também é apresentado em *"Coluna Vertebral"*, a amplitude de movimentos de cada seguimento da coluna vertebral, são eles:

1. Segmento cervical
   – Flexão: mento na fúrcula
   – Extensão: mento a 18 cm da fúrcula
   – Lateralização: 30 graus
   – Rotações: 60 graus
2. Segmento torácico
   – Rotação: 75 graus

– Lateralização: 30 graus

Obs.: a lateralização do segmento dorsal dá-se na transição dorso-lombar.

3. Segmento lombar

– Flexão: 60 graus

– Extensão: 30 graus

– Lateralização: 20 graus

– Rotações: 5 graus

A Figura 2.6 ilustra os movimentos básicos da coluna vertebral.

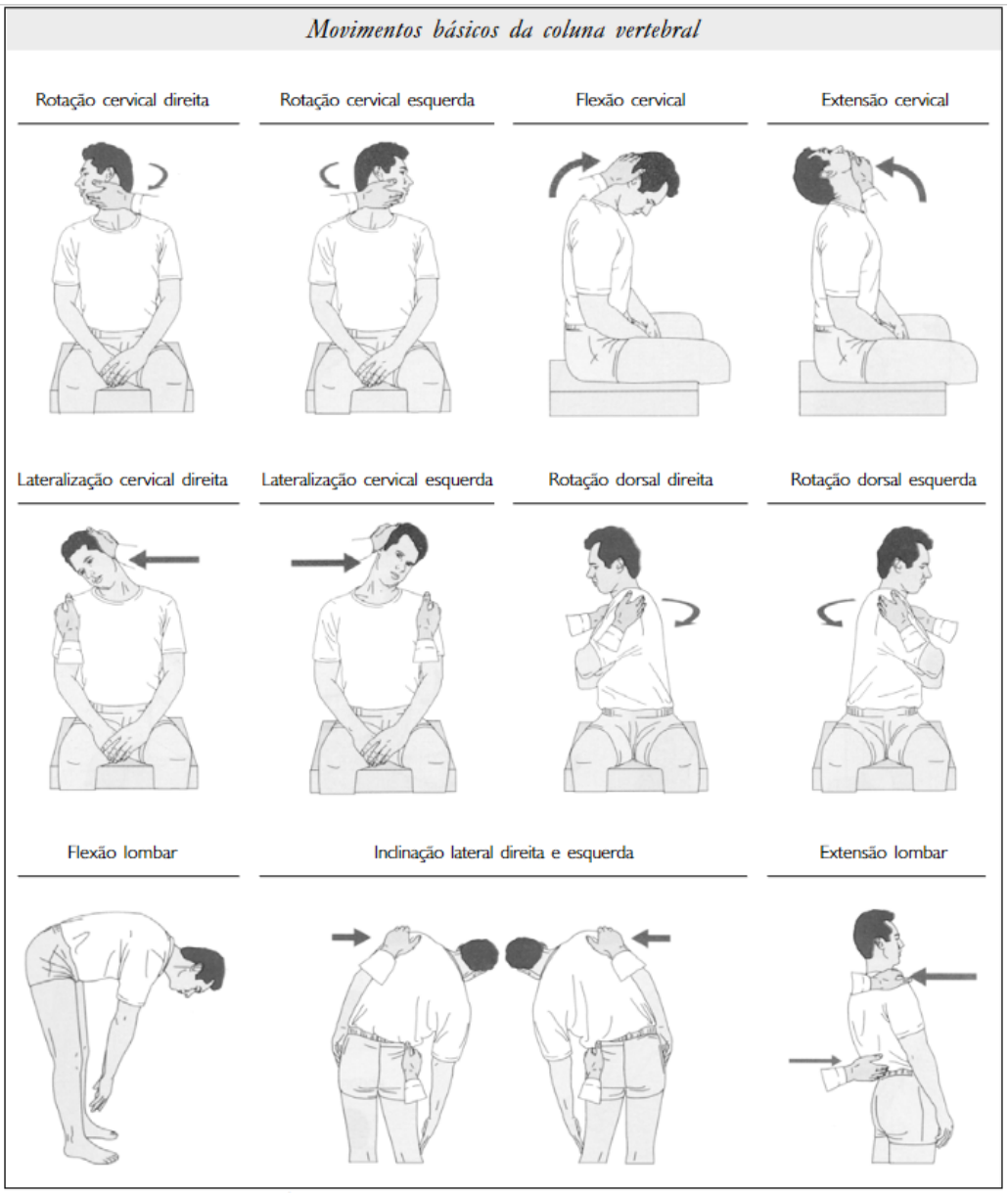

Figura 2.6 - Movimentos básicos da coluna vertebral (NATOUR et al., 2004).
Fonte: https://bvsms.saude.gov.br/bvs/publicacoes/ColunaVertebral.pdf. Acesso em 11 de set. de 2021.

2.2. Problemas relacionados à coluna vertebral

Problemas de coluna, também chamados popularmente por “dores nas costas”, englobam as cervicalgias, espondiloses, radiculopatias, transtornos dos discos intervertebrais, além de dores torácicas, lombares e ciáticas (HAGEN et al., 2002).

Cerca de 80% da população mundial irá apresentar algum tipo de queixa de dor relacionada na coluna pelo menos uma vez na vida (OMS, 2021)**.**

Desde o início da pandemia COVID-19, é observado um aumento na proporção de queixas relacionadas às dores nas costas e maior quantidade de casos crônicos relacionados à coluna. Certamente, o estilo de vida imposto pelo isolamento social contribuiu para este incremento, diz o neurocirurgião Dr. Rodolfo de Moura Carneiro, especialista no tratamento de doenças da coluna e controle da dor (DINO, 2021).

Será apresentada uma breve explicação acerca dos problemas de coluna mais recorrentes, para um melhor entendimento dos tratamentos e exercícios de reabilitação que são implementados, cujos quais serão utilizados posteriormente neste trabalho.

2.2.1 Cervicalgias

A Cervicalgia é uma síndrome caracterizada por dor e rigidez transitória, devido a movimentos bruscos dos membros superiores gera sobrecarga sobre a musculatura da coluna cervical, levando ao aparecimento de pontos gatilho, dor e tendo como consequência o declínio da atividade diária e na qualidade de vida (ANTUNES, 2017).

A prevalência de cervicalgia no âmbito da medicina ocupacional vem aumentando significativamente, sendo considerada como um dos grandes problemas da sociedade moderna (BRING et al., 1995) .

Segundo dados da OMS e em pesquisas do IBGE, o que *Bring* dizia em 1998 ainda está acontecendo e com um grande aumento nos últimos anos, as cervicalgias, assim como outras doenças que afetam a coluna vertebral, estão cada vez mais presentes nos pacientes que procuram um auxílio médico para o alívio de suas dores.

Após a obtenção de um histórico clínico completo, faz-se o exame físico no paciente. A anamnese, que é entrevista realizada pelo médico com o seu paciente durante a consulta, alerta o clínico para aqueles pacientes que se apresentam com emergências e necessitam de início precoce do tratamento, que incluem portadores de lesões expansivas causando compressões

agudas progressivas da medula espinhal e suas raízes (hérnia de disco) ou infecção (meningite bacteriana). Um exame físico bem conduzido minimiza os custos gerados pelas condições dolorosas cervicais, posto que tornam muitas vezes desnecessários exames de imagem e estudos eletrodiagnósticos (ANTONIO, 2004).

A tabela 1 apresenta as causas das dores cervicais, cujas quais são observadas durante a anamnese do paciente.

Tabela 1 - Causas de dor cervical.

Causas mecânicas
Doenças reumáticas
Infecções
Tumores e lesões infiltrativas
Doenças endócrinas, metabólicas e hereditárias
Doenças neurológicas e psiquiátricas
Dor referida
Miscelânea

Fonte: “Coluna vertebral”. p. 47, 2004.

2.2.2 Lombalgia e Lombaciatalgia

A lombalgia pode ser conceituada como uma dor de característica mecânica, localizada entre a parte mais baixa do dorso (última costela) e a prega glútea, que aparece após força física excessiva em estruturas normais ou após ação de força física normal em estruturas lesadas. A lombociatalgia surge quando esta dor se irradia para as nádegas e um ou ambos os membros inferiores (FREIRE, 2004).

A lombalgia se constitui em uma grande causa de morbidade e incapacidade, sendo apenas superada pela cefaleia, na escala de distúrbios dolorosos que afetam o homem (DEYO, 1996).

Do ponto de vista evolutivo, as lombalgias, lombociatalgias e ciáticas podem ser caracterizadas como agudas ou lumbagos, subagudas e crônicas (SHCON, 1996).

As dores lombares podem ser primárias ou secundárias, com ou sem envolvimento neurológico (BIGOS, 1991).

Inúmeras circunstâncias contribuem para o desencadeamento e cronificação das síndromes dolorosas lombares (algumas sem uma nítida comprovação de relação causal) tais como: psicossociais, insatisfação laboral (SHCON et al., 1996), obesidade (LEBOEUF et al., 1999), hábito de fumar (DEYO, 1996), grau de escolaridade, realização de trabalhos pesados (CECIN, 1992), sedentarismo, síndromes depressivas (LEINO, 1993), litígios trabalhistas (HELIOVAARA, 1991), fatores genéticos e antropológicos, hábitos posturais, alterações climáticas, modificações de pressão atmosférica e temperatura (MCGORRY, 1998).(Brazil et al., 2004).

A história clínica é essencial para avaliação diagnóstica do paciente com lombalgia e lombociatalgia. A idade do paciente poderá indicar a causa de sua dor, pois a incidência de certas doenças varia de acordo com a idade e com o sexo. Trabalho e lazer, esportes praticados, também são importantes para o diagnóstico, pois com base nos achados de Nachemson (1965, 1985), a flexão e rotação da coluna lombar aumenta a pressão no segmento motor inferior. Tem-se que as lombalgias e lombociatalgias podem ser primárias ou secundárias, com e sem envolvimento neurológico, sendo classificadas em:

1. mecânico-degenerativas;
2. não mecânicas localizadas: inflamatórias, infecciosas e metabólicas;
3. psicossomáticas;
4. como repercussão de doenças sistêmicas

(FREIRE, 2004).

2.2.3 Hérnia Discal

Hérnia discal é definida como um processo contínuo de degeneração discal que provoca a migração do núcleo pulposo além dos limites fisiológicos do ânulo fibroso (VIALLE, 2010).

As hérnias lombares são as mais frequentes, acometem principalmente o sexo masculino, na faixa de 30 a 50 anos e nos segmentos L4-L5 e L5-S1. Pela relevância epidemiológica, a doença é considerada um problema de saúde mundial, em decorrência da incapacidade física e funcional que pode gerar (BARROS, 2009)(DEYO, 1990)(STIENEN, 2011).

Dependendo do volume de material herniado, poderá haver compressão e irritação das raízes lombares e do saco dural, representadas clinicamente pela dor conhecida como ciática.

Essa dor é conhecida desde a Antiguidade, mas a sua relação com a hérnia discal não foi descoberta até o início do século 20, quando Mixter e Barr a descreveram (MIXTER; BARR, 1934).

O sofrimento da raiz nervosa não é apenas uma consequência da compressão pelo material nuclear. Sabe-se que o edema e a congestão da raiz também são fatores preponderantes no desenvolvimento dos sintomas (CORTET, 1992).

A hérnia discal pode ser confirmada na Ressonância Nuclear Magnética (RNM) ou Tomografia Computadorizada (TC) em até 30% de assintomáticos. Da mesma forma, após a cura medicamentosa da dor ciática, a imagem de hérnia em conflito com a raiz permanece inalterada por várias semanas ou meses. Tem-se evidenciado o papel da inflamação local como fator algogênico, ou seja, que reduz a temperatura do corpo (RYDEVIK, 1984) (GARFIN, 1995).

O diagnóstico de radiculalgia lombar é essencialmente clínico. Uma anamnese bem orientada associada a um bom exame físico é tudo o que o médico precisa para fazer um diagnóstico correto na maioria das vezes. O exame físico deve sempre incluir um exame geral para diagnóstico de patologias associadas, ou mesmo de causas sistêmicas de dor referida na região lombar que possam estar sendo diagnosticadas como hérnia. Exames radiológicos como TC ou RM tem como objetivo principal confirmar a origem discal de uma compressão radicular diagnosticada clinicamente (REVEL, 1992); (RADU, 2004).

2.2.4 Escolioses

A escoliose é o desvio lateral não fisiológico da linha mediana. Devido ao alinhamento vertebral e às relações estruturais das bordas vertebrais e às articulações posteriores, a inclinação lateral é acompanhada por rotulação simultânea (CAILLIET, 1979).

A mesma acontece devido a um movimento de torção generalizado por toda a raque. Esse movimento é produzido por uma perturbação localizada que origina uma ruptura do equilíbrio raquidiano (PERDRIOLLE, 1977).

Existem tipos diferentes de escoliose, de acordo com o tipo de curvatura que a coluna faz. Dentre estes podemos citar os mais recorrentes, são eles:

2.2.4.1. Escoliose Estrutural Idiopática

Responsável por aproximadamente 80% dos casos de escoliose, a escoliose estrutural é importante, progressiva e incapacitante no futuro. Existe um fator genético associado e, embora

não se evidenciem anomalias cromossômicas, aparece com maior frequência em gêmeos, irmãos e familiares (CAILLIET, 1979).

A escoliose estrutural idiopática pode ser assim dividida:

1. Infantil (0 – 3 anos)

    a) resolutiva

    b) progressiva

2. Juvenil (3 – 10 anos)

3. Adolescente (acima de 10 anos)

    (FERREIRA, 2004).

A Escoliose Idiopática Infantil é relativamente rara, sendo a incidência menor do que 1%. Quando ocorre, 90% dos casos apresentam resolução espontânea, porém 10% restantes evoluem para escoliose muito grave (MOE, 1978). Essa doença é progressiva e continua progredindo na adolescência.

Já a Escoliose Idiopática Juvenil ocorre em 12% a 21% dos casos com incidência em ambos os sexos. Trata-se de uma enfermidade nitidamente familiar, sendo frequentes as pequenas curvas não progressivas, sendo a curva torácica direita a mais comum. Não ocorre remissão espontânea, e no começo da adolescência pode evoluir de forma rápida quando não tratada (FERREIRA, 2004).

Por fim, a Escoliose Idiopática no Adolescente está presente m 2% a 4% das crianças entre 10 e 16 anos de idade. É nessa época que muitos pacientes se apresentam para diagnóstico e tratamento. Alguns já com curvas bem estruturadas e que evoluem rapidamente, cerca de 1º por mês, na fase de crescimento rápido (9 aos 13 anos). Por outro lado, existem curvas pequenas que não progridem na adolescência. É mais frequente no sexo feminino, sendo a curva torácica direita a mais comum. Sua prevalência na população geral é de 1,8%, se curvas inferiores a 10º forem incluídas. É um problema mundial; com estudos de prevalência na China, Japão, União Soviética, Suécia, Estados Unidos e Brasil mostrando notáveis resultados similares (WINTER, 1986).

2.2.4.2. Cifose Juvenil

A cifose é outra deformidade muito frequente na infância e adolescência. Muitas vezes é confundida com vício postural e assim o diagnóstico precoce é negligenciado. A postura viciosa pode ser uma manifestação de alterações morfológicas graves na coluna vertebral (FERREIRA, 2004).

Uma cifose de menos de 40º raramente tem importância estética. Entretanto, se a curva evolui, a deformidade clínica será mais acentuada, inclusive na criança obesa. As deformidades maiores de 65º são volumosas, aumentam a lordose compensadora e provocam a projeção da coluna cervical para a frente, ocasionando um aspecto estético extremamente desagradável. Curvas desta magnitude podem continuar evoluindo mesmo depois do crescimento completo. A dor nas costas pode ser transitória, com ou sem tratamento (BRADFORD, 1969).

2.2.4.3. Lordose

Hiperlordose é o menos frequente dos desvios posturais. Geralmente resulta de alterações na força, atividade ou comprimento dos músculos abdominais, espasmo dos músculos extensores da coluna ou contratura em flexão do quadril (KENDALL, 1983)(WALKER, 1987).

Raramente é o problema primário, exceto nos casos de fusão vertebral posterior congênita, após procedimento para o tratamento de hidrocefalia e em acondroplasia com estenose vertebral (MOE, 1978).

A lordose lombar, por si só, raramente requer tratamento, exceto em situações onde ela é a deformidade primária. É usualmente secundária à cifose torácica e, se não é, pode ser corrigida adequadamente com exercícios terapêuticos programados para fortalecimento da musculatura abdominal e manutenção da inclinação pélvica (WALKER, 1987).

2.2.4.4. Escoliose Degenerativa

A escoliose degenerativa é o resultado da degeneração assimétrica da coluna vertebral. O envelhecimento da população aumenta a incidência dos processos degenerativos osteomusculares, como a Escoliose do Adulto (EA).

A EA é definida como uma deformidade da coluna vertebral que apresenta um ângulo de Cobb >10° no plano coronal, em um paciente esqueleticamente maduro (AEBI, 2005). Ela pode ocorrer em razão do processo degenerativo da coluna (chamada de escoliose de *novo*),

como progressão de uma escoliose pré-existente da infância/adolescência (Escoliose Idiopática do Adulto [EIA]) ou de forma secundária a doenças sistêmicas e cirurgias prévias da coluna (YOUSSEF, 2013).

Acredita-se que a EA se inicie com a degeneração discal. Durante o envelhecimento, o disco intervertebral perde proteoglicanos pelo aumento da atividade das proteases com consequente diminuição da pressão osmótica e da hidratação discal (VERNON, 2008). Foi demonstrado que a partir dos 15 nos de idade as lesões anulares são comuns e também comprometem a biomecânica discal. Este processo resulta na perda da altura discal e na incapacidade do disco de desempenhar seu papel estabilizador, acarretando sobrecarga da coluna posterior (SILVA, 2010).

A EIA ocorre em indivíduos com escoliose preexistente, principalmente do sexo feminino, em dois padrões principais. Um grupo apresenta progressão constante após a maturidade esquelética e outro somente apresenta evolução da curva por volta da 4ª e 5ª décadas de vida, após a menopausa (MARTY, 2007).

2.3. Reabilitação

A reabilitação busca maximizar a função do paciente e reduzir a dor e a incapacidade por ele sofridas, utilizando-se tratamentos não medicamentosos (NATOUR, 2004).

Segundo o livro *Coluna Vertebral*, os objetivos da reabilitação são:

- Prevenção de disfunção;
- Restauração da função motora;
- Manutenção da função motora;
- Diminuição da dor.

O livro traz ainda que os objetivos citados podem ser alcançados através da melhoria da Amplitude De Movimento (ADM), da força, da mobilidade, das Atividades da Vida Diária (AVDs), da vida profissional e da autoeficácia, além das órteses, adaptações e educação do paciente.

Uma equipe de reabilitação deve idealmente ser composta por vários especialistas: reumatologista, fisiatra, ortopedista, fisioterapeuta, terapeuta ocupacional, nutricionista, enfermeira, assistente social e educador físico.

Muitas intervenções e métodos de reabilitação apresentam base científica mas, segundo Natour (2004), muito ainda é feito baseando-se na experiência pessoal ou da comunidade dos profissionais de saúde. A tabela 2 traz algumas evidências científicas relacionadas à reabilitação em lombalgia.

Tabela 2 - Evidências científicas sobre reabilitação em lombalgia (NATOUR, 2004)

Objeto de estudo	*Resultados do levantamento*	*Conclusões*
Exercício para lombalgia	39 ensaios controlados aleatórios	Forte evidência de que exercício não é bom para lombalgia aguda e que foi mais efetivo que fisioterapia convencional para lombalgia crônica aumentando o retorno às AVDs e ao trabalho.
Back schools para lombalgia não específica	15 Ensaios randomizados – 3 alta qualidade	Evidência moderada de melhor efeito, em curto prazo, que placebo, lista de espera em ambientes ocupacionais ou outros tratamentos (quiropraxia, exercícios). Discussão: em longo prazo? Que tipo para qual paciente? Custo-efetividade?
Tratamento comportamental	6 (25% dos estudos) de alta qualidade	Forte evidência de efeito positivo moderado sobre a intensidade da dor e positivo fraco no estado funcional geral e medidas comportamentais comparado com o controle ou lista de espera. Moderada evidência sem efeito adicional ao tratamento usual, em curto prazo, no estado funcional geral, intensidade da dor e medidas comportamentais. Que tipo de terapia comportamental e para quais pacientes?
TENS	5 ensaios com 170 sujeitos	Sem evidências que suportem o uso do TENS isolado.
Fortalecimento de extensores	21 artigos	Houve ganho de força mas sem análise de outros resultados.
Suporte lombar	33 estudos	Redução de flexo-extensão e lateralização, mas sem outros resultados clínicos.
Acupuntura para lombalgia aguda e crônica	11 estudos controlados aleatórios – apenas dois de alta qualidade	Sem estudos claramente avaliando lombalgia aguda. Sem evidência de efetividade.
Repouso		Evidência de benefício para lombalgia aguda até dois dias quando possível. Deve-se alternar com atividades.

Fonte: “Coluna vertebral”. p. 208, 2004.

Para o processo de reabilitação, a avaliação médica é essencial. Para avaliar o paciente antes do início do tratamento planejar a estratégia a ser adotada e acompanhar a evolução, medindo o impacto das intervenções realizadas, vários métodos são utilizados: medidas de ADM, força, reflexos e sensibilidade, avaliação da dor, habitualmente através de escala visual analógica de dor, avaliação funcional e de qualidade de vida, através de questionários como Roland-Morris e SF36 (NATOUR, 2004).

A coluna vertebral tem seus movimentos bem definidos e seus limites também. Para se evitar forçar a coluna além desses limites, algumas técnicas para a proteção da coluna são fundamentais. Tais técnicas evitam atitudes e maneiras de executar tarefas que sejam agressivas à coluna, podendo desencadear crises dolorosas.

Quando qualquer atividade ou exercício provoca dor, deve-se diminuir sua intensidade ou frequência, ou mesmo abandoná-lo. O uso de órteses, a orientação ergonômica e a divisão do trabalho entre os indivíduos podem diminuir a sobrecarga sobre as estruturas mais afetadas pela doença. Conselhos simples como deslizar objetos em vez de levantá-los, intercalar períodos de descanso durante o dia, não concentrar em um mesmo dia atividades mais fatigantes, entre outros, podem promover melhoria da qualidade de vida e da produtividade do indivíduo (NATOUR, 2004).

Também é utilizado o repouso como medida de reabilitação, que pode ser sistêmico ou em uma região. O repouso diminui a inflamação, a dor e a contratura, mas se prolongado demais enrijece estruturas, compromete a integridade da cartilagem, diminui a capacidade cardiopulmonar, a massa óssea e a massa muscular, além de gerar distúrbios emocionais, as vezes de difícil tratamento. Sendo assim, o tratamento deve ser feito o mais cedo possível para se evitar o estado de cama dos pacientes.

Para os exercícios que são realizados, podem ser passivos ou ativos, sendo estes divididos em três categorias:

- Exercícios isométricos;
- Exercícios isotônicos;
- Exercícios isocinéticos.

Os exercícios indicados pelo fisioterapeuta devem ter continuidade com uma atividade física adequada à situação clínica de cada paciente.

O tratamento com exercícios é o mais utilizado e apresenta melhores resultados, mas existe também outros tratamentos, como o medicamentoso, que pode ser utilizado juntamente com exercícios fisioterápicos, e o tratamento cirúrgico.

O tratamento medicamentoso é baseado em analgésicos, anti-inflamatórios não esteroidais, anticonvulsivantes e antidepressivos (DEPALMA, 2008).

As principais indicações cirúrgicas são quando da limitação em atividades de vida diária, dor, sintomas neurológicos, confirmação da progressão da curva e falha do tratamento conservador (HALDEMAN, 2018).

Os pacientes que apresentam os melhores resultados com tratamento clínico são aqueles com deformidades leves, não progressivas e pouco sintomáticos (PASSIAS, 2018).

Na reabilitação, além dos exercícios, existem meios e equipamentos que auxiliam no processo de tratamento dos pacientes, são eles:

- Meios físicos;
- Órteses;
- Adaptações para atividades do dia a dia;
- Educação do Paciente.

2.4. Jogos Sérios

A educação nunca esteve tão relacionada ao entretenimento como está hoje: aprender e ensinar de forma lúdica se tornou objeto de pesquisa e estudos, e atualmente já é uma realidade. Foi em 2002 que se deu início ao movimento de Jogos Sérios, que se espalhou rapidamente graças à *Serious Games Initiative* (Susi, Johannesson e Backlund, 2007).

Entende-se Jogos Sérios como aplicações que mesclam aspectos sérios como o ensino, a aprendizagem, a comunicação e a informação, com o lúdico e interativo fornecido pelos videogames, sendo o primeiro o principal objetivo e não somente o entretenimento (Michael e Chen, 2006); (Alvarez e Djaouti, 2011).

Certos videogames prontos (edições comerciais COTS – *Commercial Off-The-Shelf*) também podem ser utilizados para propósitos sérios (Alvarez e Djaouti, 2011; Squire et al., 2005). Todavia são os Jogos Sérios que possuem o objetivo principal e inicial de servir a um propósito sério, em especial, a educação.

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 2.7 mostra a ligação entre os jogos sérios e os jogos comuns e como é o desenvolvimento dessa relação.

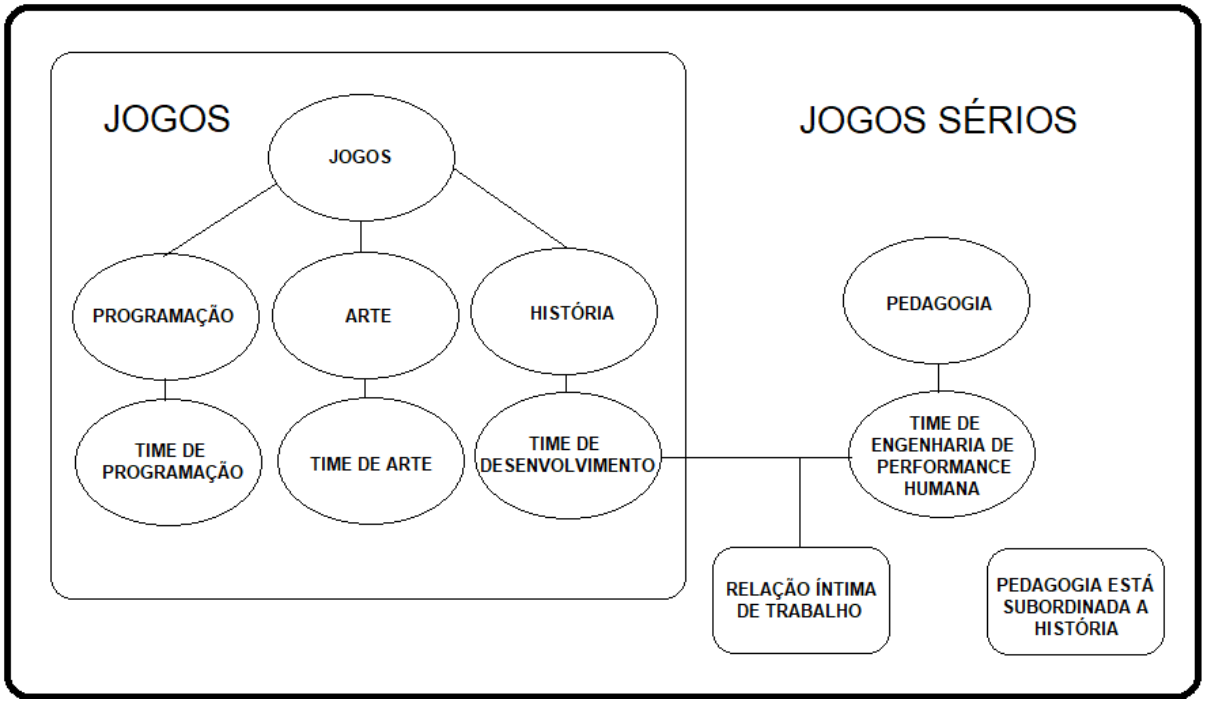

Figura 2.7 - Desenvolvendo jogos sérios através dos jogos comuns (ZYDA, 2005) (modificado).

2.5. *OpenPose*

O OpenPose foi o primeiro sistema em tempo real de código aberto para detecção de pose 2D de várias pessoas, incluindo corpo, pé, mão e pontos chave faciais.

É de autoria de Ginés Hidalgo, Zhe Cao, Tomas Simon, Shih-En Wei, Yaadhav Raaj, Hanbyul Joo e Yaser Sheikh e é mantido por Ginés Hidalgo e Yaadhav Raaj (HIDALGO et al., 2019). O OpenPose possui aplicação em diversas áreas, dentre as quais o campo da saúde/reabilitação pode se tornar amplamente utilizado.

O OpenPose possui um Plugin Unity para a utilização da ferramenta na criação e desenvolvimento de jogos. OpenPose Unity Plugin é um *wrapper* da biblioteca OpenPose para usuários Unity. Ele fornece saída OpenPose formatada e alguns exemplos. Este projeto é de autoria de Tianyi Zhao, Gines Hidalgo e Yaser Sheikh. Atualmente, ele está sendo mantido por Tianyi Zhao e Gines Hidalgo (HIDALGO et al., 2019).

As figuras a seguir ilustram exemplos do OpenPose Unity Plugin sendo utilizado.

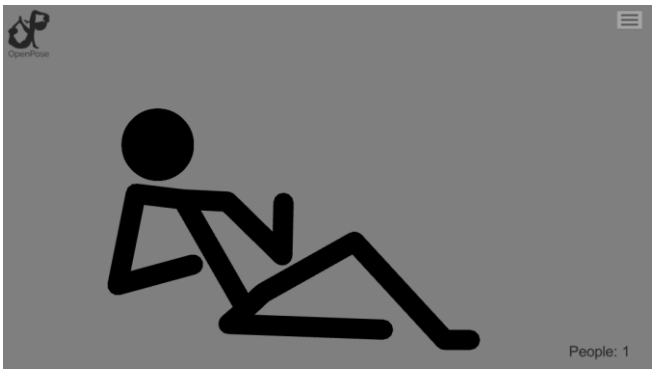

Figura 2.8 - Teste do plugin OpenPose Unity para detecção de corpo e pés.

Figura 2.9 – Test do plugin OpenPose Unity para detecção de corpo, pé, rosto e mãos.

Figura 2.10 - Tianyi Zhao (esquerda) e Gines Hidalgo (direita) testando o plug-in OpenPose Unity.
Fonte: https://github.com/CMU-Perceptual-Computing-Lab/openpose_unity_plugin. Acesso em 10/09/2021.

Os desenvolvedores do OpenPose realizaram uma análise de tempo de execução entre o OpenPose e outras 2 bibliotecas de estimativas de pose disponíveis e amplamente utilizadas, sendo a análise realizada com um mesmo hardware e nas mesmas condições para todos os testes (HIDALGO et al., 2019).

As três bibliotecas utilizadas foram:

- OpenPose;
- Alpha-Pose (FANG et al., 2017);
- Mask R-CNN (CLAUSS, 2017).

O resultado da análise apresentou que o tempo de execução do OpenPose é constante, enquanto o tempo de execução do Alpha-Pose e Mask R-CNN cresce linearmente com o número de pessoas.

A figura 2.8 mostra graficamente o resultado descrito.

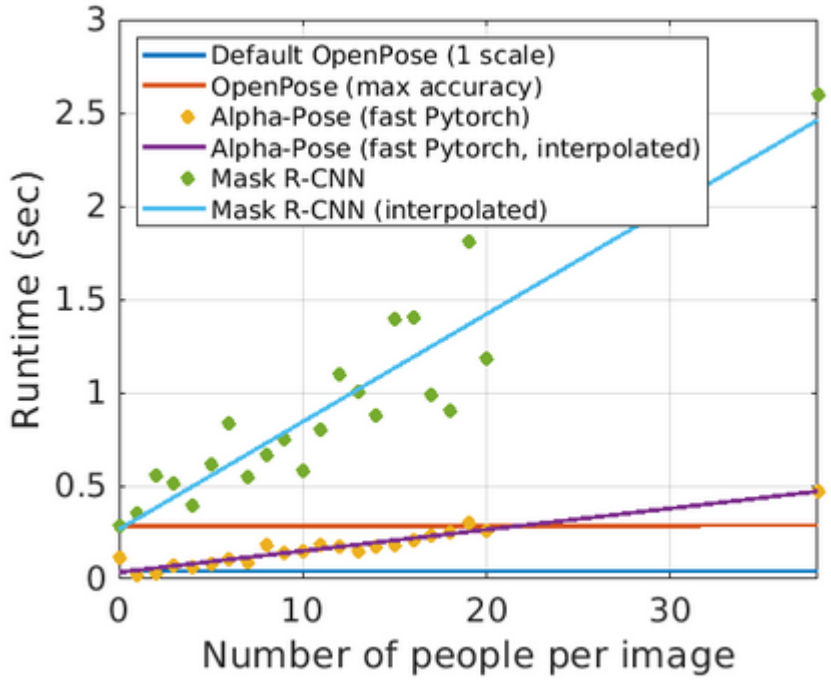

Figura 2.11 - Análise do tempo de execução entre as 3 bibliotecas analisadas.
Fonte: https://github.com/CMU-Perceptual-Computing-Lab/openpose#readme. Acesso em 10 set. 2021.

A eficiência e acurácia do OpenPose é citada no artigo "*OpenPose: Realtime Multi-Person 2D Pose Estimation using Part Affinity Fields*", que fala do desempenho ao se utilizar um conjunto de campos vetoriais 2D que codificam a localização e orientação dos membro do corpo humano sobre o domínio da imagem (CAO et al., 2019).

É demonstrado que inferir simultaneamente representações ascendentes de detecção e associação [método utilizado pelo OpenPose] codifica o contexto global, gerando uma análise com resultados de alta qualidade (CAO et al., 2019).

A Figura 2.12 mostra o método de análise de imagem que o OpenPose realiza. (a) O método usa a imagem inteira como entrada para um CNN (*Convolutional Neutral Network*) para prever em conjunto (b) mapas de confiança para detecção de partes do corpo e (c) PAFs (*Part Affinity Fields*) para associação de partes. (d) A etapa de análise realiza um conjunto de correspondências bipartidas para associar candidatos de parte do corpo. (e) Finalmente estes são montados em poses de corpo inteiro para todas as pessoas na imagem.

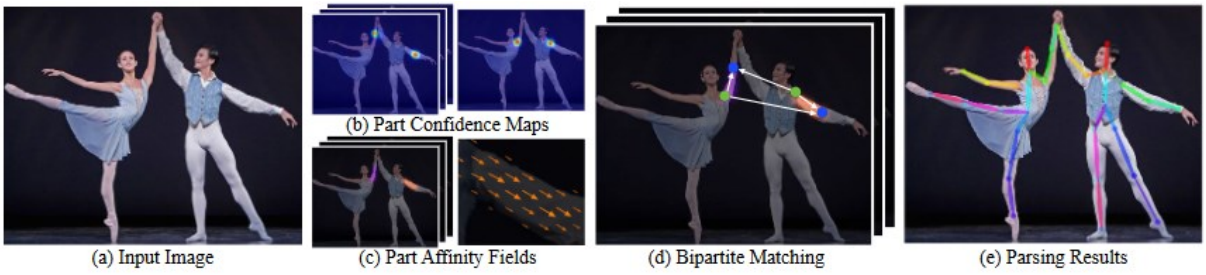

Figura 2.12 – Método de análise de imagem utilizada pelo OpenPose.
Fonte: https://arxiv.org/pdf/1812.08008.pdf. Acesso em 10 set. 2021.

MACEDO e SANTOS dizem em seu trabalho "*Análise Cinemática Automática usando OpenPose e Dynamic Time Warping com Aplicações no Remo*", de 2019, que o maior diferencial do OpenPose está na sua capacidade de fazer estimação de múltiplas pessoas de maneira conjunta e precisa, sem afetar significamente o tempo de processamento. Segundo os mesmos, isto ocorre pois, ao invés de realizar a estimação individualmente para cada pessoa identificada, o método realiza parte dos procedimentos em paralelo (MACEDO e SANTOS, 2019).

Em "*Human Action Recognition Based On Spatiotemporal Features From Videos*", SILVA cita que o OpenPose alcança bons resultados na extração de poses 2D com velocidade de processamento em tempo real e em sequências de vídeo irrestritas. Através da extração de pose 2D utilizando o OpenPose, o método consegue reconhecer ações humanas em vídeo sequências em tempo real, com excelente precisão e com ótimo desempenho (SILVA, 2020).

Portanto, esse projeto de pesquisa tem como objetivo desenvolver jogos controlados pelo OpenPose que possam auxiliar profissionais da saúde na fisioterapia de pacientes com problemas de dores na coluna vertebral.

2.6. Unity 3D

A produção e mercado de entretenimento de jogos digitais mostra um grande crescimento no mundo atual. Independentemente da plataforma, em sua natureza, um jogo digital é um software interativo. Para produzir qualquer tipo de software, normalmente, utiliza-se alguma ferramenta que possa facilitar o trabalho. A ferramenta usada no auxílio ao desenvolvimento de jogos é chamada de motor de jogo (TORI, 2006).

A Unity 3D é um motor de jogos (*game engine*) amplamente conhecido e utilizado para criação de jogos 2D e 3D. Os jogos desenvolvidos através da Unity podem ser utilizados, com as devidas licenças, em diversas plataformas, como PC, MAC, iPhone, console Wii da Nintendo e até mesmo embutidos em uma página Web (PASSOS et al., 2009).

Com diversos níveis de aplicações e sendo uma ferramenta bastante completa, o Unity 3D tem outra característica que chama muito a atenção, que é o fato de possuir uma versão gratuita para o desenvolvimento de jogos que, embora tenha determinadas limitações, foi utilizada de maneira muito satisfatória para a execução deste projeto (LANGE et al., 2011).

2.6.1. Interface

O Unity 3D possui uma interface simples e agradável tendo o objetivo de facilitar o desenvolvimento de jogos de diversos gêneros, além de outros sistemas de visualização. Sua área de trabalho trás várias janelas chamadas *views*, onde cada uma delas tem seu propósito específico. A Figura 2.13 mostra uma representação esquemática e a identificação de cada uma dessas janelas no editor de cenas da Unity 3D.

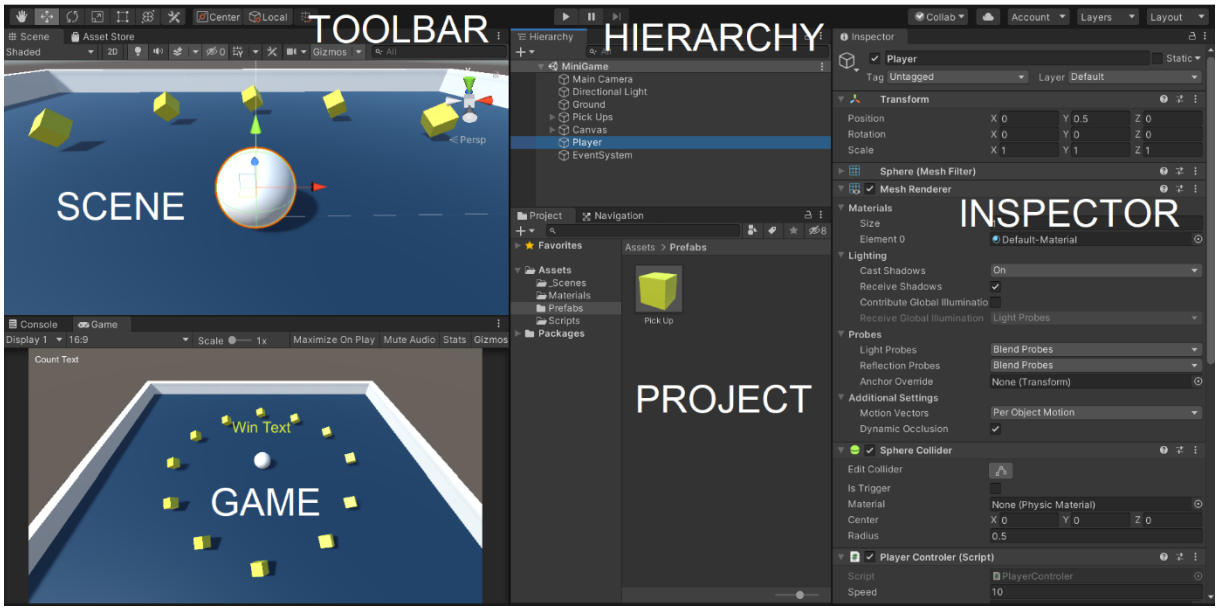

Figura 2.13 – Interface do editor de cena da Unity 3D e suas principais *views* (Unity 2019.4.24.f1).

A janela Project, representada na Figura 2.14, é a interface para manipulação e organização dos vários arquivos (Assets) que compõem um projeto tais como scripts, modelos, texturas, efeitos de áudio e Prefabs. Sua principal função é garantir a integridade dos arquivos do jogo em desenvolvimento, pois sempre que um arquivo é modificado dentro da plataforma Unity 3D este sempre será analisado pelo todo através do motor de jogos (PASSOS et al., 2009).

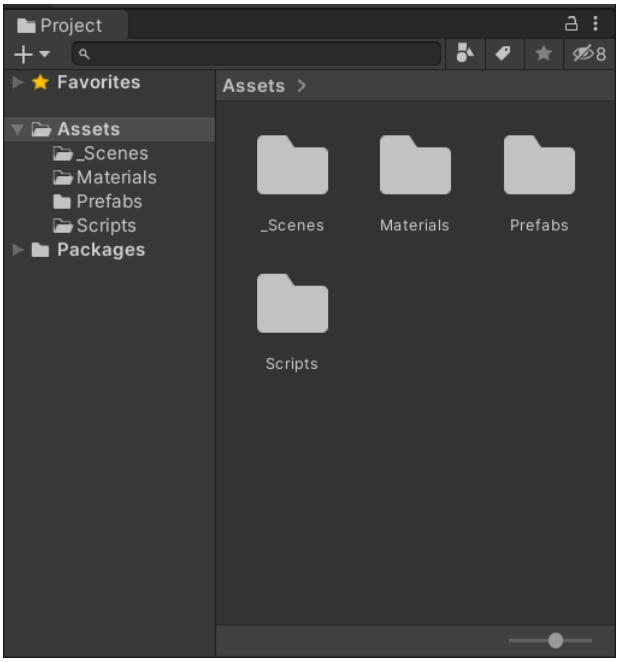

Figura 2.14 – Janela *Project*.

A janela *Hierarchy,* representada na Figura 2.15, exibe todos os elementos da cena que se está editando. Além disso, nessa janela podemos organizar e visualizar a hierarquia de composição entre os vários objetos que compõem a cena (grafo de cena). O funcionamento desses objetos, bem como a hierarquia de transformação será explicado mais detalhadamente na próxima seção.

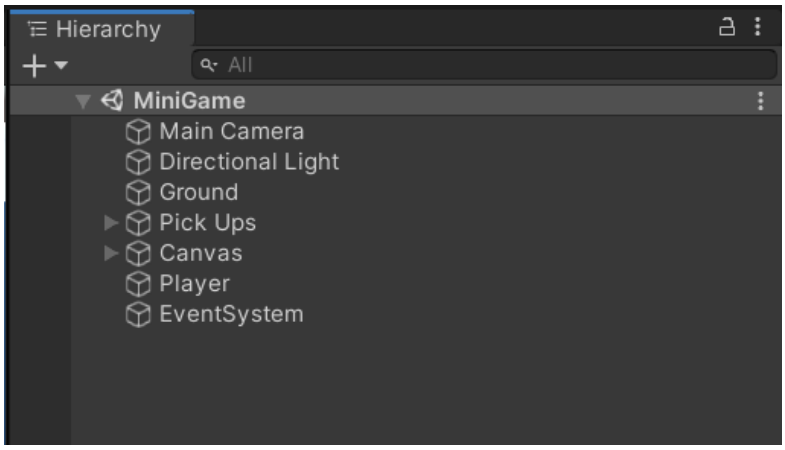

Figura 2.15 – Janela *Hierarchy*.

A janela *Scene*, representada na Figura 2.16, é a forma principal de manipulação dos elementos visuais no editor de cenas da Unity, possibilitando a orientação e posicionamento desses elementos com um feedback imediato do efeito das alterações efetuadas. Nesta janela, pode-se manipular graficamente os objetos através das opções de arrastar e soltar com o mouse. Essa manipulação é semelhante àquela de ferramentas de modelagem 3D e pode-se manipular objetos tais como câmeras, cenários, personagens e todos os elementos que compõem a cena.

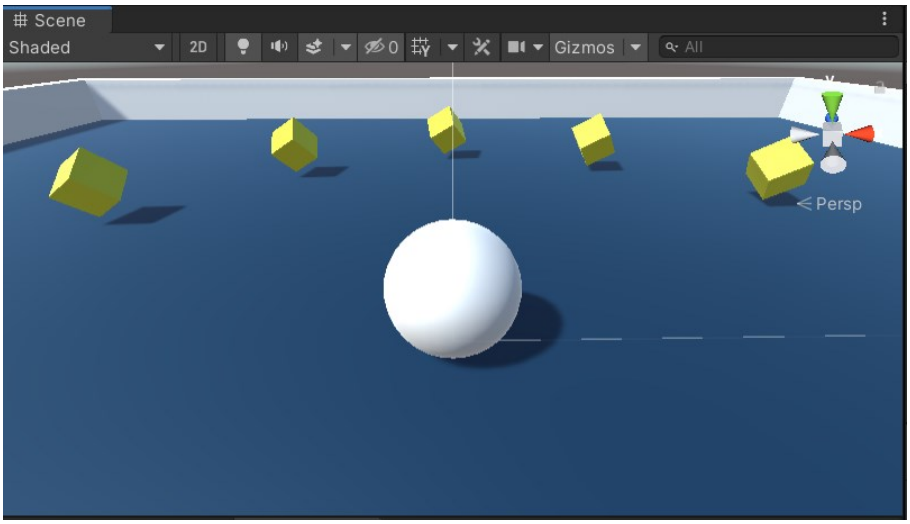

Figura 2.16 – Janela *Scene*.

A janela *Game*, representada na Figura 2.17, é responsável pela visualização da aplicação em desenvolvimento na forma que ela será exibida quando finalizada. Nessa janela, pode-se rapidamente ter uma prévia de como os elementos estão se comportando dentro da aplicação, além de se observar através dela às mudanças de parâmetros que possam ser feitas na janela de inspeção (*Inspector)*.

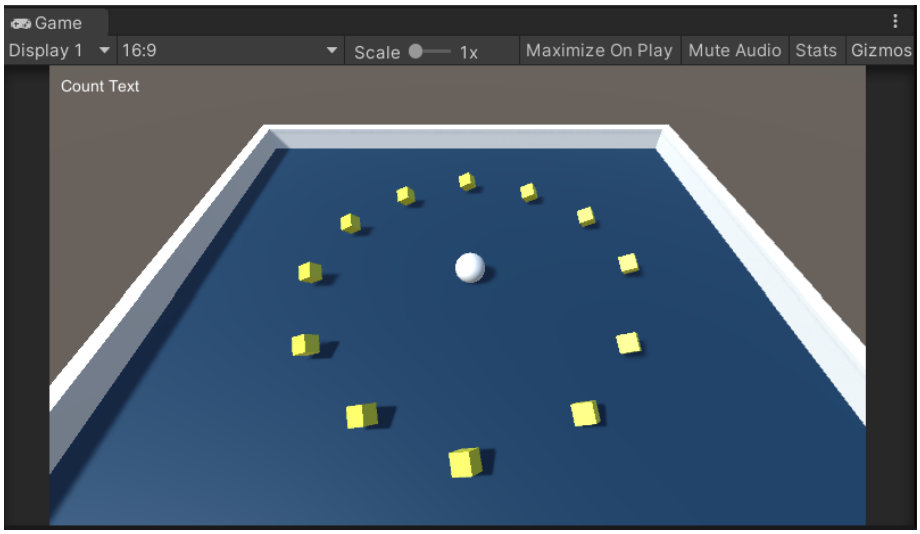

Figura 2.17 – Janela *Game*.

Na janela *Inspector*, representada na Figura 2.18, tem-se acesso aos vários parâmetros de um objeto presente no cenário, bem como aos atributos de seus componentes (*Components*). Ainda na janela Inspector, pode-se ajustar os atributos públicos (parâmetros) de cada componente, inclusive durante a execução da aplicação (PASSOS et al., 2009).

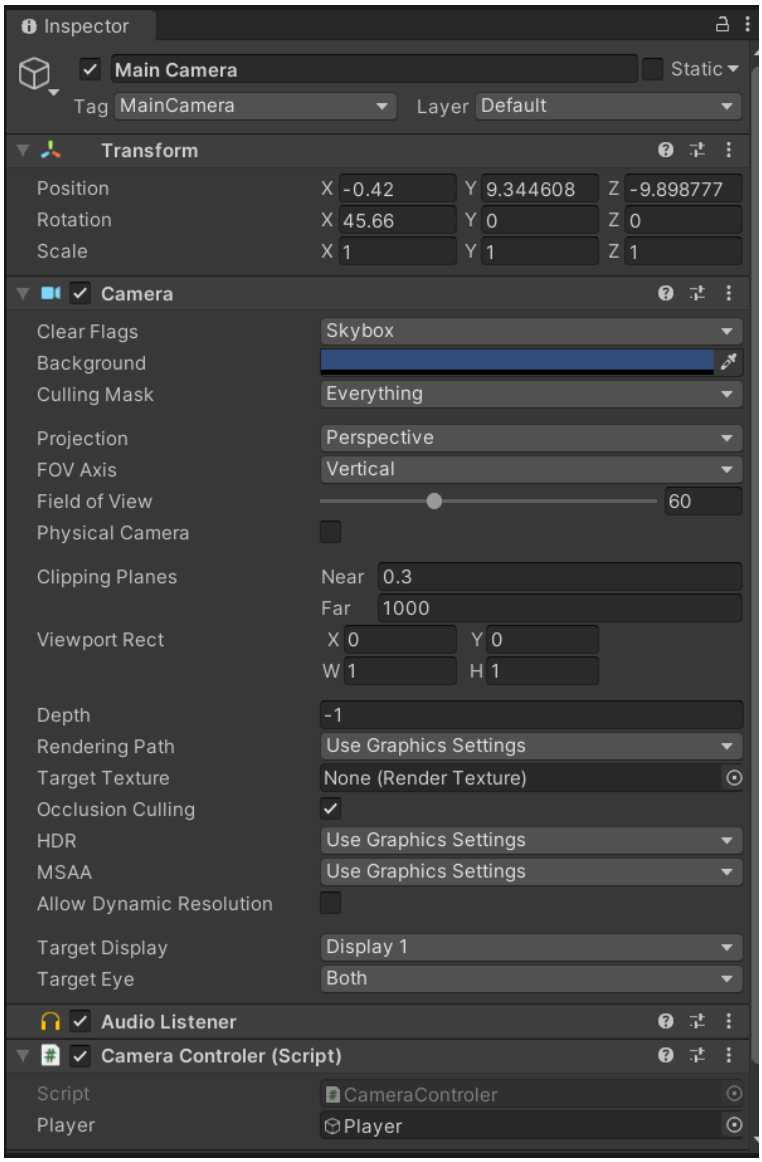

Figura 2.18 – Janela *Inspector*.

Já a região denominada por *Toolbar*, representada na Figura 2.19, tem as mesmas funções e características de qualquer outra barra de ferramentas. Nela encontra-se caminhos e atalhos para diversas tarefas, além de possuir o sistema de ativação da simulação do jogo em desenvolvimento.

Figura 2.19 – *Toolbar*.

2.6.2. Importação de assets e packages

Assets são muito utilizados no desenvolvimento de jogos, são desenvolvidos em ferramentas externas e são importados para dentro do editor de cenas do motor de jogos. A Unity possui uma forma simples e robusta de importação dos mesmos para dentro de um projeto, bastando que sejam inseridos dentro de uma pasta da janela *Project*. Um *package* é o conjunto de vários *assets* que podem ou não trabalhar entre si.

Durante o desenvolvimento desde projeto foram utilizados vários tipos de *assets*, disponibilizados gratuitamente na plataforma da Unity.

Foram utilizados também outros *assets* que foram obtidos através de *downloads* na *Internet*, bem como imagens utilizadas como background e personagens utilizados nos jogos e no menu de calibração.

Além dessa simplicidade de importação dos *assets*, o Unity também oferece a possibilidade da visualização em tempo real de qualquer alteração feita nos mesmos. Com isso, tem-se um ganho de produtividade, pois não precisa-se importar manualmente novas versões para dentro do Unity a cada vez que desejarmos efetuar uma alteração. O Unity verifica cada arquivo modificado e automaticamente atualiza o mesmo na cena.

2.6.3. Desenvolvimento de jogos

Por ser uma plataforma ampla e de fácil utilização, o Unity 3D atrai os mais diversos tipos de profissionais, para as mais variadas atividades, desde artistas visuais até programadores de carreira. Sendo assim, o desenvolvimento de jogos no Unity se dá de diversas maneiras e com variadas técnicas de criação dos jogos.

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

2.6.3.1. Sistema de GameObjects

O Unity3D é baseada em um modelo moderno para a arquitetura de objetos de jogo baseado em composição (Bilas 2002; Stoy 2006; Passos et al. 2008). Nesse modelo, um objeto de jogo é especificado através da composição de várias funcionalidades, que são agregadas (ou removidas). Cada funcionalidade é implementada por um componente (classe que herda de um componente básico). Esse container genérico é denominado GameObject e funciona como um repositório de funcionalidades, ou mais especificamente, componentes (Passos et al. 2009).

Os componentes são então responsáveis por implementar os diversos comportamentos que um GameObject pode ter. Um componente pode ser desde um script, uma geometria de

colisão, ou até uma textura. Ou seja, GameObjects podem representar qualquer coisa no cenário, sendo caracterizados como uma simples câmera ou um personagem apenas pelos diferentes componentes que agrega. Conforme observado no manual de usuário do Unity (tradução livre): "*GameObject é uma panela vazia e os componentes são os ingredientes que irão criar sua receita de jogabilidade*" (PASSOS et al., 2009).

Analisando os GameObjects nota-se que todos possuem, no mínimo, o componente *Transform*, onde está definida sua posição, orientação e sua escala do jogo. É utilizado para definir hierarquias, sendo possível o acesso a objetos pais e/ou filhos através do componente *Transform*. Tais ações podem ser acompanhadas pela janela *Inspector.*

A Figura 2.20 mostra a janela Inspector do objeto de jogo "Pescador". Analisando a imagem, é possível se observar o componente *Transform.* Além do componente *Transform*, nota-se um componente de Script chamado "Spawnpescador", onde nele se tem o código responsável por controlar as ações do Player "pescador1".

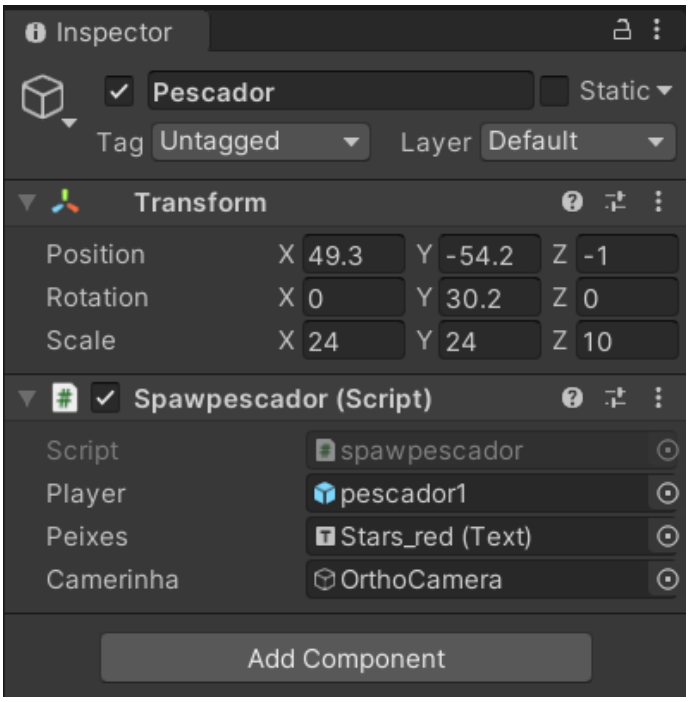

Figura 2.20 – GameObject "Pescador".

Mais abaixo na Figura 2.20, é possível se observar um botão intitulado "*Add Component*". Este botão é responsável pela adição de componentes dentro dos objetos de jogo.

A Figura 2.21 ilustra alguns tipos de componentes capazes de serem adicionados a um objeto de jogo.

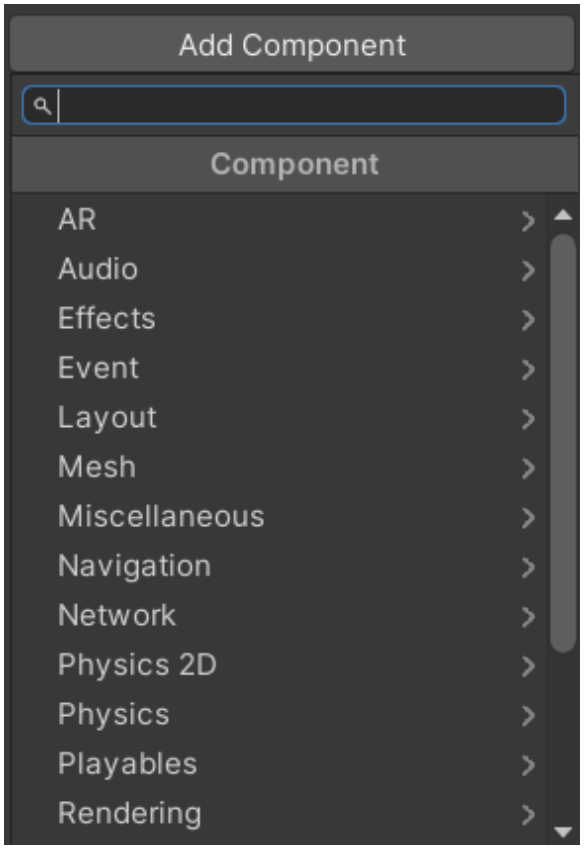

Figura 2.21 – Exemplos de componentes a serem adicionados a um objeto de jogo.

A partir dos conceitos de objeto de jogo e através de componentes é possível criar objetos pré-definidos, os chamados *Prefabs* (modelos pré-fabricados).

Um *Prefab* é simplesmente um modelo de composição de GameObject já definido, ou mais precisamente, um *template* que define um elemento através da composição dos vários componentes (PASSOS ET AL., 2009).

2.6.3.2 Scripting

O sistema de *Scripting* do Unity3D é abrangente e flexível, o que permite o desenvolvimento de jogos completos sem a necessidade do uso de C/C++. Internamente, os scripts são executados através de uma versão modificada da biblioteca *Mono*, uma implementação de código aberto para o sistema *.Net*. Essa biblioteca permite que os scripts sejam implementados em qualquer uma de três linguagens, à escolha do programador: *Javascript*, *C#* (C *Sharp*) ou *Boo* que é um dialeto de Python (PASSOS et al., 2009).

Para o projeto em questão, a linguagem escolhida para o desenvolvimento dos *scripts* dos jogos foi a C#, devido à proximidade com a linguagem C e por ter a grande presença na literatura pesquisada durante o projeto.

Através dos *scripts* é possível ter acesso a outros componentes do objeto de jogo que contém o *script*. Ainda através de variáveis públicas, é possível obter informações de outros objetos de jogo instanciados na mesma cena.

Também é possível acessar informações da hierarquia do objeto de jogo por meio do componente *Transform,* utilizando técnicas de *scripting*.

A Figura 2.22 mostra a tela inicial de um *script* em sua forma padrão inicial feito no *Visual Studio 2017*.

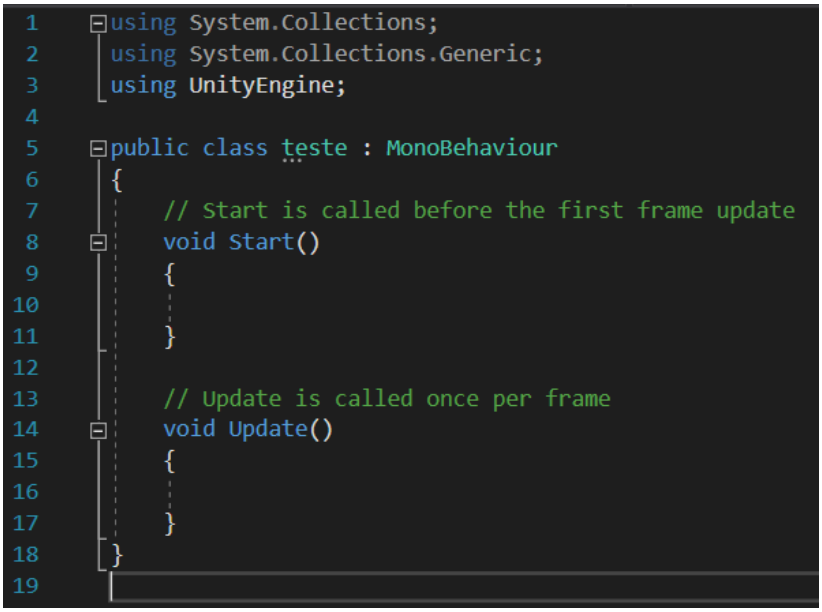

Figura 2.22 – Exemplo de código padrão inicial do *script* no software *Visual Studio 2017*.

CAPÍTULO III

DESENVOLVIMENTO

Neste capítulo serão apresentadas as metodologias utilizadas, bem como e as etapas de desenvolvimento do projeto, além das devidas justificativas.

3.1. Escolha dos exercícios fisioterápicos

Inicialmente foi decidido quais seriam os tipos e a quantidade de exercícios fisioterápicos para se basear os jogos. Foram escolhidos 6 exercícios, divididos em 4 jogos.

Os exercícios escolhidos foram:

- Extensão lombar (método Mckenzie), posição deitado;

Figura 3.1 – Extensão Lombar (método Mckenzie), posição deitado.
Fonte: Bezerra, Anne de Souza. Uma revisão acerca dos métodos: Mckenzie vs.Williams. 2016, p. 8.

- Extensão lombar (método Mckenzie), posição ereta;

Figura 3.2 - Extensão lombar (método Mckenzie), posição ereta
Fonte: Bezerra, Anne de Souza. Uma revisão acerca dos métodos: Mckenzie vs.Williams. 2016, p. 8.

- Flexão do quadril (exercícios de Williams), posição deitado;

Figura 3.3 - Flexão do quadril (exercícios de Williams), posição deitado.
Fonte: Bezerra, Anne de Souza. Uma revisão acerca dos métodos: Mackenzie vs.Williams. 2016, p. 8.

- Flexão lombar, posição ereta;

Figura 3.4 – Flexão lombar, posição ereta.
Fonte: https://www.researchgate.net/figure/Figura-13-A

- Flexão lateral esquerda;

Figura 3.5 – Flexão lateral esquerda.
Fonte: https://www.easyvigour.net.nz/fitness/h_LumbarSideBend.htm

- Flexão lateral direita;

Figura 3.6 – Flexão lateral direita.
https://www.easyvigour.net.nz/fitness/h_LumbarSideBend.htm

O Método Mckenzie é uma técnica de avaliação e de auto tratamento da Fisioterapia Avançada baseada em evidência, que oferece condições para o paciente se tratar de maneira rápida, segura e eficaz, sem depender de medicação, calor, gelo, ultrassom ou cirurgias. Esta abordagem é conhecida no mundo inteiro como Método Mckenzie de Diagnóstico e Terapia Mecânica (em inglês, Mechanical Diagnosis and Therapy – MDT). Tal método de tratamento utiliza tanto exercícios de extensão quanto de flexão (BEZERRA, 2016).

Já os exercícios de Williams foram desenvolvidos observando-se que a maioria dos pacientes que apresentavam dores lombares crônicas, possuíam alterações degenerativas esqueléticas secundárias a lesões dos discos intervertebrais. Sendo assim, utiliza como princípio do tratamento, exercícios de flexão da coluna e quadril. Com o propósito de reduzir a dor e estabilizar o tronco, desenvolve ativamente os músculos flexores e alonga passivamente os músculos extensores lombosacros. Williams dá muito enfoque na questão da inclinação posterior da pelve sendo essencial para obter ótimos resultados no tratamento (BLACKBURN e MCKENZIE 1981).

Outros exercícios foram escolhidos inicialmente, porém devido algumas limitações encontradas, e que serão discutidas na seção “Resultados e Discussões”, não foi possível a utilização deles neste projeto. Sendo assim, separou-se os 6 exercícios escolhidos em 4 jogos com a divisão apresentada a seguir. Os jogos desenvolvidos serão detalhados na seção “Jogos”.

Jogo 1 - Asteroides:

- Extensão lombar (método McKenzie), posição deitado.

Jogo 2 – Dia de Parque:

- Flexão lombar, posição ereta;
- Extensão lombar (método McKenzie), posição ereta.

Jogo 3 - Pesca:

- Flexão do quadril (exercícios de Williams), posição deitado.

Jogo 4 – Corrida Infinita:

- Flexão lateral esquerda;
- Flexão lateral direita.

3.2. Modelagem dos jogos

Os jogos foram modelados de maneira que o profissional de saúde, responsável pelo paciente e com conhecimento na área de terapia e reabilitação, supervisiona e acompanha as etapas dos jogos, ou seja, desde a entrada de dados do paciente até a finalização do jogo.

Sendo assim, a figura 3.7 mostra o diagrama de utilização entre usuário fisioterapeuta, usuário paciente e os jogos.

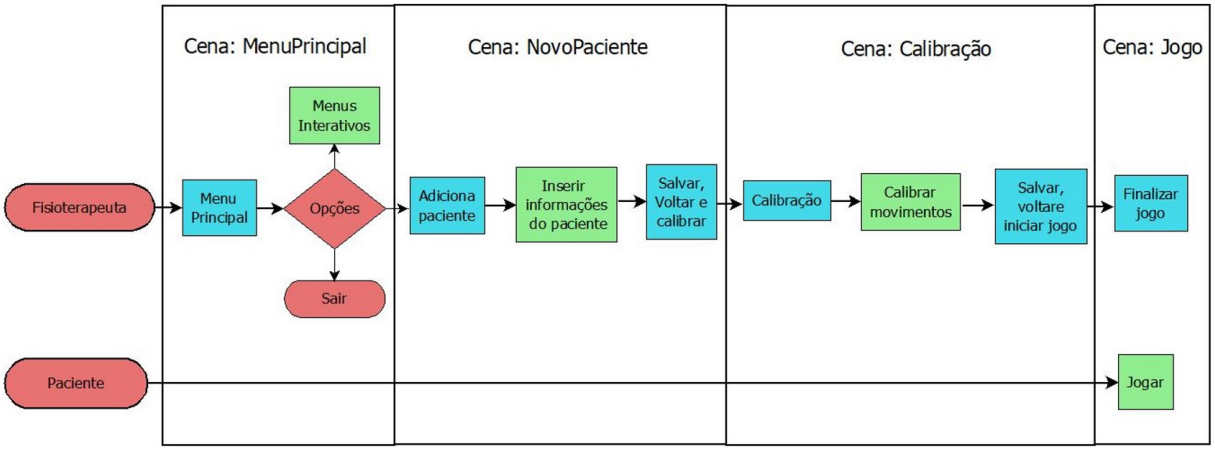

Figura 3.7 – Diagrama de utilização dos jogos.

Na figura 3.7 é possível observar como é a interação entre os usuários, terapeuta e paciente, e os jogos. Observa-se as 4 telas principais e o objetivo de cada uma, onde, cada tela possui uma função definida e a sequência de funcionamento pré-estabelecida.

No menu principal o usuário/fisioterapeuta terá acesso à cinco caminhos que pode tomar: **cadastro**, **calibração**, **jogos**, **opções** e **sair**. Na guia opções, apenas será mostrado um texto informativo sobre o jogo, não tendo a necessidade de interação com o usuário. Já na opção sair, o usuário será encaminhado para fora do menu e fora dos jogos.

A opção "cadastro" levará o usuário para a segunda tela, a cena "NovoPaciente", como mostra a figura 3.7. Esta tela permite ao usuário inserir um novo paciente no banco de dados do menu, através da inserção de informações do paciente pelo usuário.

Com as informações do paciente inseridas, a próxima opção do usuário é realizar a calibração dos movimentos, destinados à seção fisioterápica atual, por parte do paciente, onde seus limites de movimento serão registrados para utilização no controle de movimentação nos jogos. Tal calibração é feita na cena "Calibração".

Realizada a calibração dos movimentos do paciente, a opção será de iniciar o jogo escolhido para a seção fisioterápica atual, previamente escolhida e planejada pelo fisioterapeuta responsável.

Nesta tela de ambiente do jogo, quem irá interagir com o jogo é o usuário paciente, que será representado pelo avatar de player principal de cada jogo. O mesmo terá os objetivos de cada jogo sempre relacionados aos movimentos de coluna que o tratamento do paciente exige, onde efeitos visuais aparecem em forma de "premiação" a cada acerto de movimento, e permanência naquela posição durante o tempo necessário, que o paciente conseguir realizar.

Também há, nesta tela, um mostrador de pontos e de tempo de partida para cada jogo, bem como um menu de pause do jogo, a ser utilizado exclusivamente pelo usuário terapeuta. Neste menu, encontrado em cada um dos jogos, o usuário terapeuta tem acesso a quatro caminhos que pode tomar: **continuar**, **menu principal**, **salvar** e ***quit***. Na opção continuar, o usuário paciente apenas é retornado ao jogo para a sua continuidade.

Já na opção "menu principal", o usuário terapeuta é confrontado com uma pergunta, onde é indagado se tem certeza de tal escolha, podendo escolher entre duas opções, **sim** e **não**. Caso escolha a opção "sim", o jogo é finalizado e o usuário terapeuta é levado de volta ao menu principal sem salvar os dados de jogo, que são a "pontuação" e "tempo de partida". Caso escolha a opção "não", o usuário terapeuta é encaminhado de volta ao menu de jogo.

Outra opção existente neste menu é a "salvar", onde o usuário terapeuta salva os dados de jogo do paciente, os já citados "pontuação" e "tempo de partida". Tais dados são adicionados ao arquivo gerado na cena "NovoPaciente", ilustrado na Figura 3.7, pré-existente, como duas novas linhas de dados ao final do arquivo. Esta opção não apresenta efeitos visuais, apenas salva os dados e o usuário terapeuta continua no menu do jogo.

A última opção do menu de jogo é a "*quit*", onde o usuário terapeuta é confrontado, novamente, com uma pergunta, onde é indagado se tem certeza de tal escolha, podendo escolher entre duas opções, **sim** e **não**. Caso escolha a opção "sim", o jogo é finalizado sem salvar os dados de jogo. Caso escolha a opção "não", o usuário terapeuta é encaminhado de volta ao menu de jogo.

Escolhendo a opção de retornar ao menu principal, o usuário terapeuta pode navegar nas opções deste normalmente, ou sair do menu e dos jogos conforme já foi explicado anteriormente.

3.3. Os jogos

Nesta seção serão apresentados todos os aspectos dos jogos, de forma a ambientar melhor o leitor com os mesmos e com os seus desenvolvimentos. As informações aqui apresentadas serão complementadas com os documentos informados nos Apêndices e Anexos deste trabalho.

3.3.1. Interface menu principal

Ilustrada na figura 3.8, a interface do menu principal apresenta uma imagem de fundo e cinco botões que executam as ações nomeadas em cada um deles.

Figura 3.8 - Interface da cena “MenuPrincipal”.

Para a criação de uma tela como a apresentada na Figura 3.8, deve-se utilizar elementos de usuário do Unity 3D, a *UI System*. A partir destes componentes é possível criar telas de interação com o usuário a partir de textos, botões, controles de alternância, controles deslizantes, entre outros.

Tais elementos devem ser criados dentro de um objeto de jogo denominado *Canvas* para uma melhor organização, onde se tornam objetos filhos do mesmo, assim como apresentado na Figura 3.9.

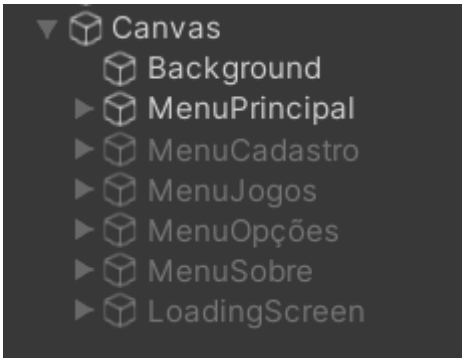

Figura 3.9 – Representação do objeto de jogo do Tipo *Canvas* dentro da janela *Hierarchy*.

Na Figura 3.9 é possível se observar os cinco elementos filhos do painel *Canvas*. O elemento *MenuPrincipal* é o menu principal, aquele apresentado na Figura 3.8. Este elemento é o painel padrão ativo, ou seja, será a tela mostrada quando a cena do menu principal for ativada, que acontece quando o jogo é inicializado ou quando se retorna a ele através do menu de jogo. Já o elemento *Background* é apenas um *GameObject* contendo a imagem de fundo do painel.

O elemento *MenuCadastro* é a tela onde são inseridas as informações referentes ao paciente, pelo usuário terapeuta. Este elemento será detalhado na seção *3.3.2. - Interface de inserção de Paciente*.

O próximo elemento, denominado *MenuJogos*, é a tela onde são apresentados os quatro jogos disponíveis para se realizar a sessão de fisioterapia do paciente. Cada jogo tem sua aplicabilidade com exercícios específicos e cabe ao usuário fisioterapeuta decidir qual é o exercício a ser realizado naquela sessão específica. Este elemento é mostrado na Figura 3.10 e será melhor detalhado na seção *3.3.4 – Interface de seleção dos jogos*.

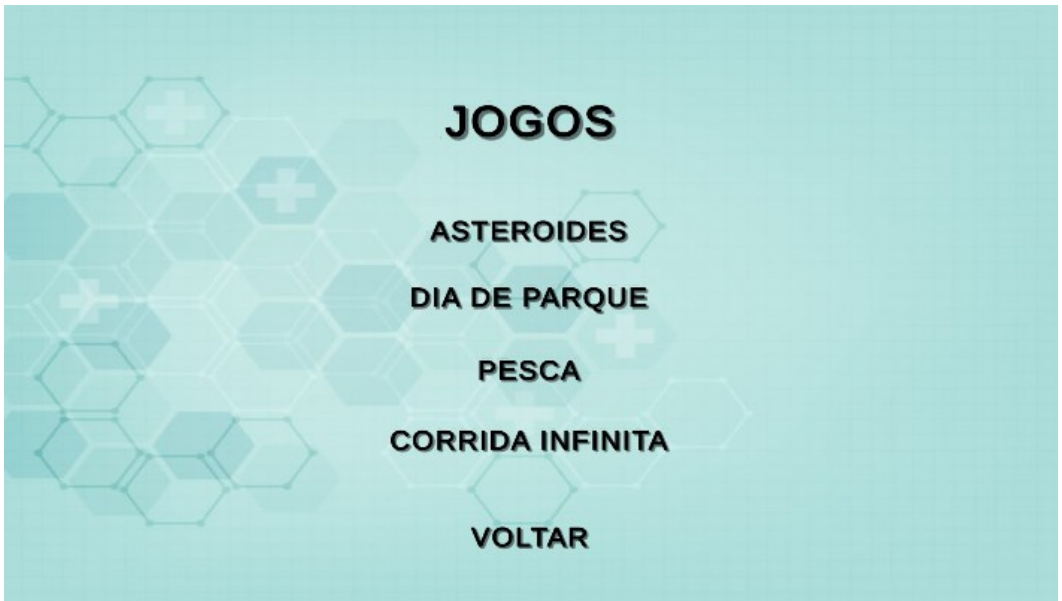

Figura 3.10 - Interface do elemento *MenuJogos*.

Os jogos apresentados no elemento *MenuJogos* serão detalhados na seção *3.3.5. – Detalhando os jogos*.

O próximo elemento é o *MenuOpções*, onde estão as configurações possíveis de serem realizadas pelo usuário terapeuta dentro do menu, que são alterar o volume da música e ter acesso a informações sobre o jogo e seu desenvolvimento.

Figura 3.11 - Interface do elemento *MenuOpções.*

A opção "Sobre", apresentada no *MenuOpções* da Figura 3.11, abre o elemento "*MenuSobre*" onde são apresentadas as informações de desenvolvimento dos jogos. A Figura 3.12 ilustra a interface principal deste elemento.

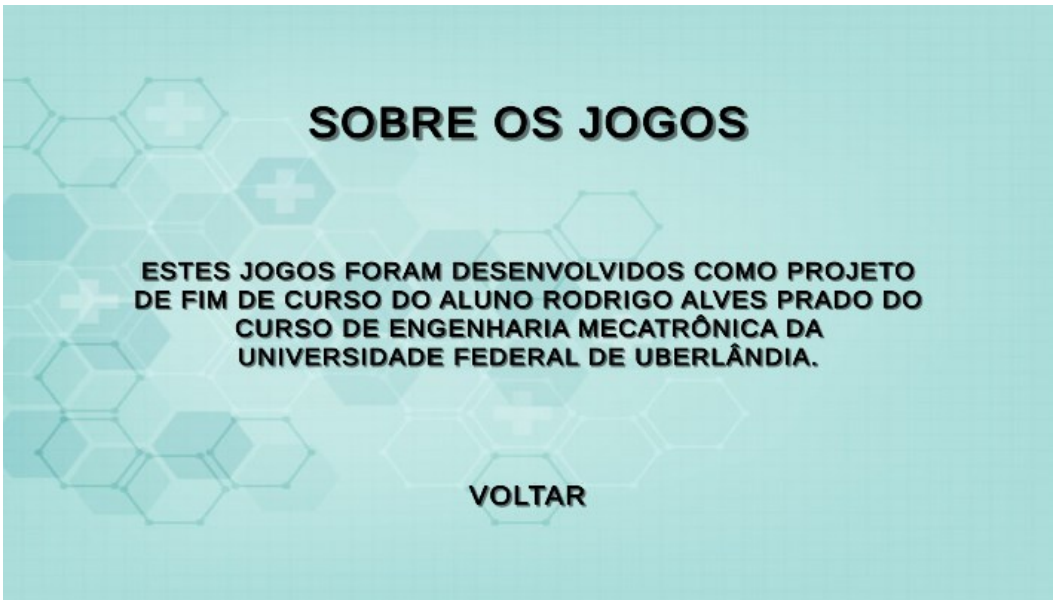

Figura 3.12 – Interface da tela "Sobre os Jogos".

O último elemento contido no painel *Canvas* é o *LoadingScreen*, o qual é a tela de carregamento entre cenas. Nela é apresentada uma barra de carregamento com a porcentagem da cena que já foi carregada, até seu completo carregamento e inicio da mesma. A Figura 3.13 mostra esta cena.

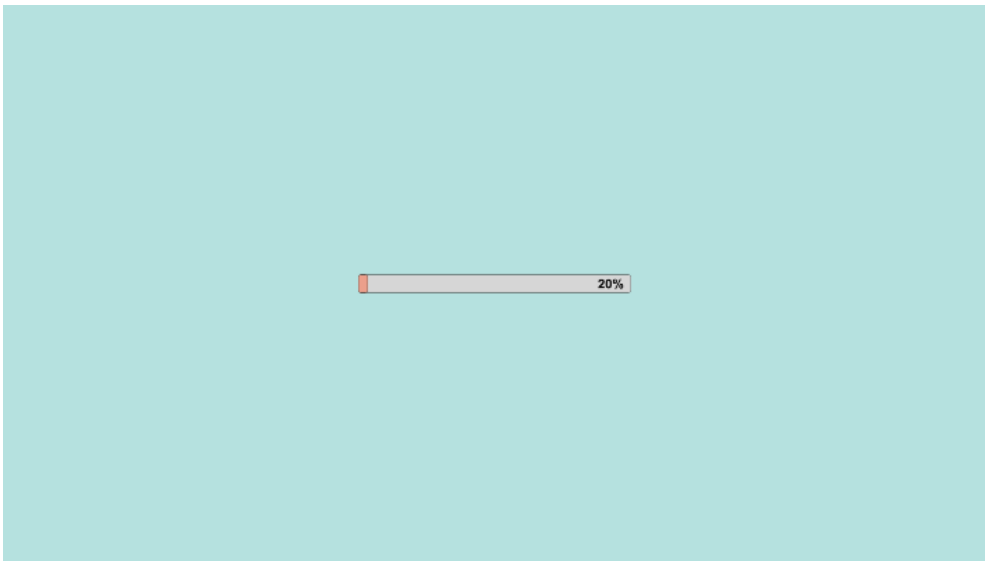

Figura 3.13 – Tela do elemento *LoadingScreen*.

Na cena *MenuPrincipal* estão contidos os objetos apresentados na Figura 3.14. Todos estes são objetos de jogo com componentes específicos para cada um deles.

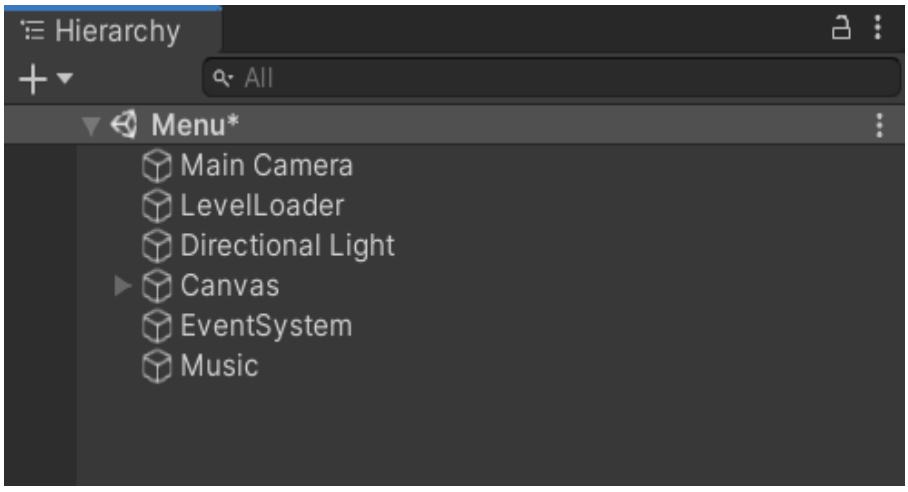

Figura 3.14 - Janela *Hierarchy* mostrando os objetos de jogo da cena atual.

Cada um dos elementos apresentados na Figura 3.14 será rapidamente detalhado. Todos estes são objetos de jogos, como dito anteriormente, com componentes específicos que os tornam este tipo de objeto.

O primeiro objeto observado é o *Main Camera* que 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, onde seu principal componente é o Transform que foi modificado para que fosse mostrado a área de interesse do cenário.

Já o *LevelLoader* é chamado de objeto vazio. Este objeto vazio é um objeto de jogo como qualquer outro, porém, fisicamente, ele não representa nada para o jogo. Este tipo de objeto é utilizado para criar pontos de referência para a inserção de scripts relacionados a cena em questão.

Para este caso em questão, o LevelLoader possui o script "*LevelLoader.cs"* que é responsável por obter a porcentagem de carregamento para a próxima cena e com estas informações passar os valores necessários para a tela *LoadingScreen*. Tal script pode ser completamente visualizado no Apêndice D.

O objeto *Directional Light* é um objeto de jogo responsáveis pela iluminação da cena. Este possui 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.

Já 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 *Music* é um elemento vazio, assim como o elemento *LevelLoader*. É utilizado 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 de jazz instrumental neutro e suave. Isto fará com o usuário paciente fique calmo e relaxado até se iniciar o jogo escolhido.

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.

3.3.2. Interface de inserção de paciente.

Nesta cena, o usuário terapeuta insere as informações do funcionário através de um formulário interativo. As informações inseridas serão salvas em um arquivo de texto em uma pasta previamente selecionada pelo profissional. A Figura 3.15 mostra a visão geral desta cena.

Figura 3.15 - Visão geral da cena de inserção de paciente.

Esta cena é composta por apenas uma tela e foi feita separadamente para um melhor controle de gravação dos dados do paciente.

Ao se clicar no botão “Cadastro” da cena “MenuPrincipal”, o jogo carrega esta cena. Com a cena carregada, o usuário terapeuta preenche os dados do paciente com o teclado do computador.

As informações solicitadas neste formulário são:

1. Nome;
2. Idade;
3. Gênero;
4. Terapia Nº;
5. Inicio Tratamento;
6. Data Hoje;
7. Caminho do Arquivo

Tais informações são inseridas em caixas de texto posicionadas como filhas do painel *Canvas*, como acontece na cena “MenuPrincipal“, juntamente com dois botões, “salvar” e “voltar”.

O controle destas informações é feito através do *script* “*Salvararquivo.cs*”, que pode ser completamente visualizado no apêndice E.

Com este *script*, após serem declaradas todas as variáveis, cria-se um método para salvar os dados em um arquivo. Neste método, o caminho para salvar o arquivo é criado através dos campos “Caminho do Arquivo”, “Nome” e “Terapia Nº”. Os nomes dos arquivos são padronizados como “Nome - Sessão#Número da Terapia”. Seu conteúdo é mostrado na Figura 3.16.

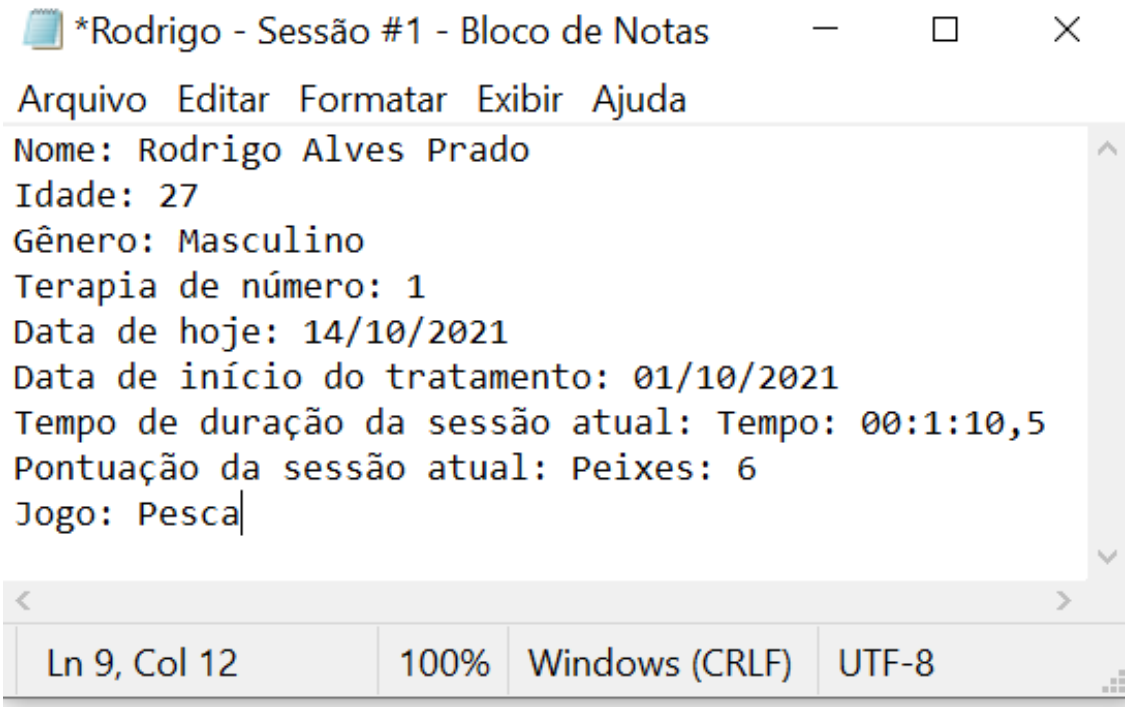

Figura 3.16 - Exemplo de relatório gerado após cada partida.

Assim como na cena anterior foi criado um objeto vazio, chamado *User*, para que fosse possível a utilização deste *script* e que o mesmo ficasse ativo em toda a cena esperando apenas o usuário clicar em “salvar”.

Não há, nesta cena, proteção contra o usuário no sentido de falta de preenchimento ou preenchimento incorreto. Apenas foi colocado nesta cena um Debug que diz que o nome do paciente não foi preenchido, assim não salvando os dados e utilizando as configurações padrões para executar a próxima tela retornando para a tela da cena “MenuPrincipal”.

3.3.3. Interface principal do “PauseMenu”

Esta cena está presente em todos os quatro jogos. É carregada na tela de jogo ao ser pressionada a tecla “*Esc*” do teclado por parte do usuário terapeuta, caso seja necessário pausar a seção de jogo. Ao ser chamada à tela, esta cena apresenta um menu interativo, com o fundo quase totalmente transparente e mostrando a tela de jogo no momento da execução em que foi pausada.

Como já foi citado no item 3.2 estão contidos neste menu quatro opções em que o usuário terapeuta pode navegar e escolher dependendo de qual ação ele pretende fazer naquele momento. São elas: **continue**, **menu principal**, **salvar** e ***quit***. A Figura 3.17 ilustra tal cena com seus componentes em um fundo fora de jogo.

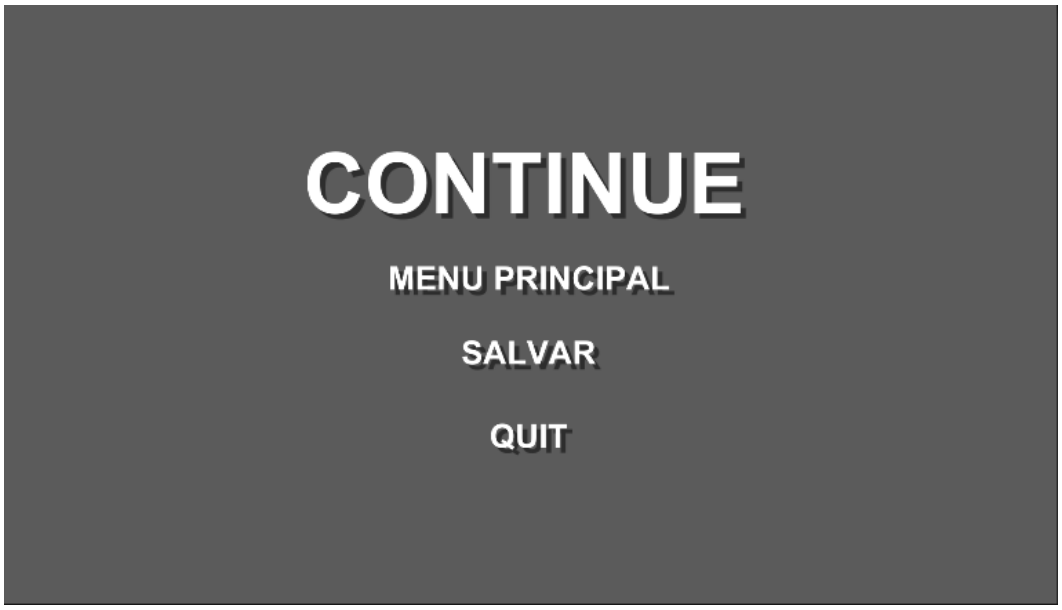

Figura 3.17 - Tela principal do "*PauseMenu*".

A Figura 3.18 mostra esta mesma cena dentro de um dos jogos, pausado durante sua execução. O jogo em questão é o “Pesca”, que será detalhado posteriormente.

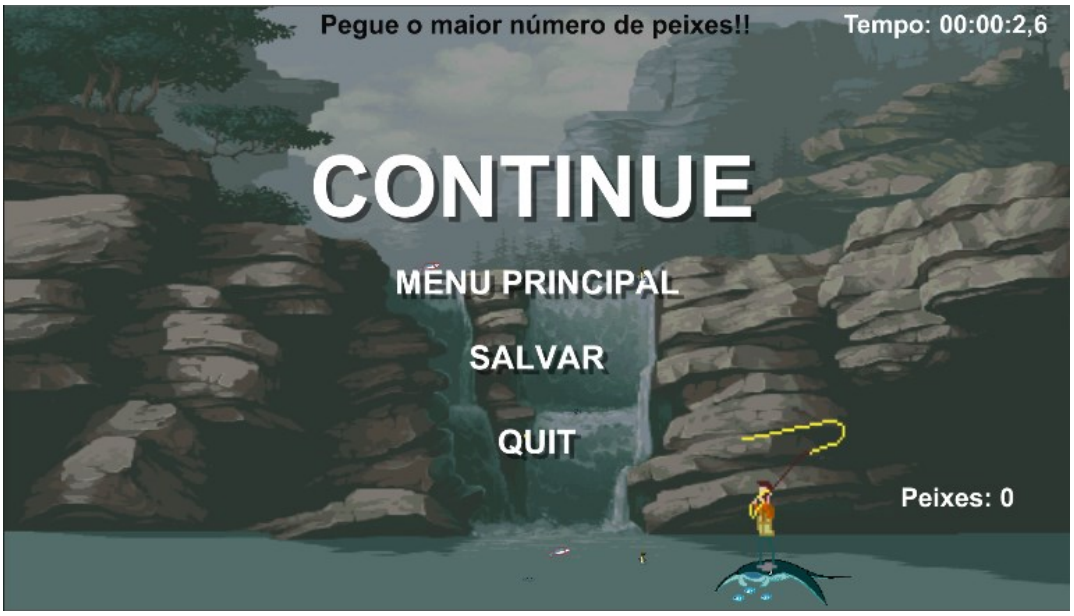

Figura 3.18 - Tela principal do "*PauseMenu*" com um jogo em execução.

As opções são ativadas ao se clicar no botão correspondente o qual vai executar a ação descrita no mesmo.

Para a criação de uma tela como a apresentada nas Figuras 3.17 e 3.18 deve-se utilizar elementos de usuário do Unity 3D, a *UI System*, assim como apresentado no item *3.3.1 – Interface MenuPrincipal*.

Como já citado, tais elementos devem ser criados dentro de um objeto de jogo denominado *Canvas* para uma melhor organização se tornando objetos filhos deste painel. A Figura 3.19 mostra o painel *Canvas* para o *PauseMenu*.

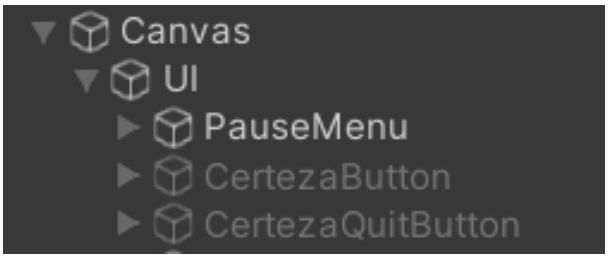

Figura 3.19 - Painel Canvas do *PauseMenu*.

Na Figura 3.19 é possível se observar que painel *Canvas* possui o elemento *UI* como seu filho e outros três elementos filhos deste *UI*. O elemento *PauseMenu* é o menu principal de pausa, aquele apresentado nas Figuras 3.17 e 3.18. Este elemento é o painel padrão ativo, ou seja, será a tela apresentada ao se ativar a cena do menu pausa, que acontece quando o usuário terapeuta apertar a tecla "*Esc*" do teclado. Nele estão contidas as quatro opções citadas anteriormente: continuar, menu principal, salvar e *quit*. O botão "salvar" salva a pontuação do jogador até o momento, bem como o tempo de execução do jogo, no mesmo arquivo utilizado para salvar o cadastro do paciente que está realizando a sessão em questão através do *script* "SalvarTempoPontuacao" disponível no Apêndice F.

Caso o usuário escolha a opção "Menu Principal", uma tela de questionamento é mostrada na sequência com duas opções, sim e não. Esta nova tela é o elemento *CertezaButton*, onde o usuário é indagado se tem certeza de que quer ir para o menu principal, clicando em "sim", e sair do jogo atual, uma vez que, caso o usuário não tenha salvo a pontuação e tempo de jogo através do botão "salvar" anteriormente, o jogo será encerrado sem salvar os devidos dados da sessão. Caso clique em "não", o usuário é levado de volta à tela principal do *PauseMenu*.

Já caso o usuário escolha a opção "*quit*", será mostrado o último elemento do *PauseMenu*, denominado *CertezaQuitButton*, que assim como no elemento *CertezaButton*, o usuário é questionado se tem certeza de que quer sair do jogo clicando em "sim" e sair do jogo atual, uma vez que, caso o usuário não tenha salvo a pontuação e tempo de jogo através do

botão “salvar” anteriormente o jogo será encerrado sem salvar os devidos dados da sessão. Caso clique em “não” o usuário é levado de volta à tela principal do *PauseMenu*.

As Figuras 3.20 mostra o jogo “Pesca” pausado, evidenciando a tela principal dos elementos *CertezaButton* e *CertezaQuitButton*.

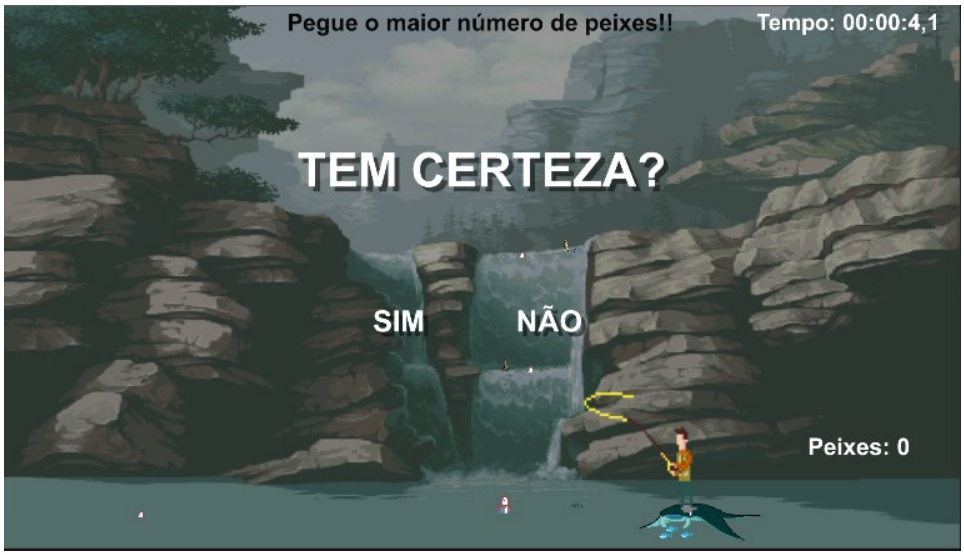

Figura 3.20 – Tela principal dos elementos *CertezaButton* e *CertezaQuitButton*.

3.3.4. Interface de seleção dos jogos

Nesta cena são apresentados ao usuários os jogos disponíveis para a realização da sessão fisioterápica atual. Quem realizará tal escolha é o profissional de saúde responsável pelo tratamento do paciente em questão.

A Figura 3.21 mostra a tela principal desta cena.

Figura 3.21- Tela referente a cena *MenuJogos*.

Esta cena consiste em um menu interativo onde o usuário terapeuta pode navegar entre as opções de jogos disponíveis e escolher um deles clicando no botão com o referido nome do jogo. Há nesta tela, também, um botão intitulado “voltar”, onde, ao ser pressionado, levará o usuário de volta à tela *MenuPrincipal*.

No momento que o usuário escolher o jogo e clicar no botão correspondente, a cena do mesmo será carregada e o usuário será direcionado para ela, iniciando, assim, a sequência de exercícios por parte do paciente.

A imagem de fundo utilizada nesta tela é mesma padrão que aparece na cena *MenuPrincipal* e nos demais menus interativos pré-jogo.

3.3.5. Detalhando os jogos

Nesta seção será apresentado o detalhamento de cada um dos quatro jogos, com suas particularidades e elementos utilizados.

3.3.5.1. Jogo “Asteroides”

Aqui será apresentada a interface principal do jogo “Asteroides”, bem como seus elementos e funcionamento.

3.3.5.1.1. Interface principal do jogo “Asteroides”

Nesta cena é apresentado o primeiro jogo em execução. Aqui o paciente irá interagir com o personagem principal do jogo através dos movimentos capturados pela câmera, seja ela uma *Webcam* ou uma outra câmera auxiliar conectada e configurada junto ao computador e processados pelo *OpenPose*.

Assim como as demais cenas principais dos outros três jogos, esta cena é muito elaborada, uma vez que necessita de grandes quantidades de processamento e controle para sua correta execução. Inicialmente será explicado sobre a interface em geral e posteriormente mais especificamente sobre os elementos que a compõem.

A Figura 3.22 (a) mostra a tela padrão desta cena. Nela está o personagem controlável pelo jogador através do *OpenPose*, via câmera, cujo qual é uma espaçonave em um cenário que representa uma batalha contra asteroides no espaço sideral. Estes asteroides vão surgindo na tela com o passar do tempo, em posições pré-definidas.

(a)

(b)

Figura 3.22 – (a) Tela principal do jogo "Asteroides". (b) Tela principal com o jogo em execução.

É possível se observar também na Figura 3.22 (a), na parte superior central, uma frase explicando o que o jogador deve fazer neste jogo, como se fosse um tutorial.

Com o jogo em execução, é possível se observar nesta cena, no canto superior direito, o placar de pontuação do jogo e o tempo de execução do mesmo, bem como um dos asteroides a ser destruído. A Figura 3.22 (b) ilustra tal cena com o jogo em execução.

O plano de fundo da tela principal está contido em um objeto 3D denominado *Quad*, cujo qual contém, também, o *script* "*CenarioInfinito.cs*" responsável por movimentar a imagem de fundo a uma velocidade previamente escolhida. Este *script* está disponível no Apêndice G.

3.3.5.1.2. Avatar do jogo "Asteroides"

O avatar é o personagem principal do jogo, tem a função de simular para o ambiente virtual os movimentos da coluna do usuário paciente, o qual estará fazendo os exercícios fisioterápicos em frente a câmera. Este personagem é um modelo pré-fabricado (*Prefab*) disponibilizado gratuitamente na internet para download por meio de *assets*.

Ele representa uma espaçonave sem articulações e responde ao movimento de dois pontos específicos da coluna, sendo um deles, localizado no pescoço do paciente, responsável por efetuar os movimentos do personagem de fato. O outro ponto, localizado na parte central do quadril do paciente, é utilizado como referência para verificar se o paciente está corretamente posicionado para realizar o exercício em questão, que é o exercício de extensão lombar (método McKenzie), realizado com o paciente deitado no chão com a parte frontal do corpo em direção ao chão. O modelo utilizado pode ser visualizado na Figura 3.23 (b).

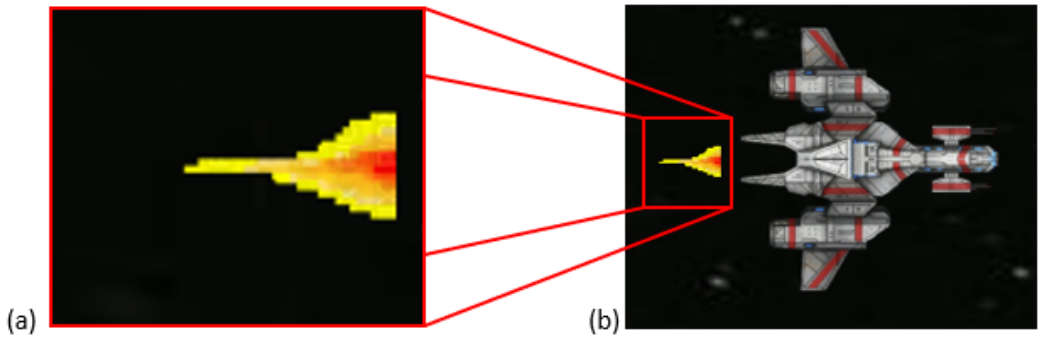

Figura 3.23 – (a) Jato propulsor da nave em detalhe. (b) Modelo utilizado para o avatar do jogo "Asteroides".

Este é um modelo simples, constituído apenas de um *sprite*, se comporta como se fosse uma imagem na tela, que é movimentada através do *script* "*Cruiser.cs*", cujo qual está disponível para visualização no apêndice H. Este *script* faz com que o personagem se movimente entre três posições específicas, a fim de cumprir o proposto pelo exercício fisioterápico. As posições são obtidas através da posição da coluna do paciente em dois pontos pré-estabelecidos, marcadas em quatro variáveis do tipo "*PlayerPrefs*", na tela de calibração. Tendo uma variável responsável pelo valor de coordenada "*x*" e uma pelo valor "*y*" para cada um dos dois pontos. Estes pontos são tidos como o limite superior e limite inferior de movimentação da coluna.

Tais pontos são interpretados através do *script Cruiser.cs* e são acrescidos a estes valores uma margem de posicionamento, tanto para mais, quanto para menos, em que ao movimentar a coluna, o paciente tenha visualmente a confirmação que está se posicionado corretamente

durante o exercício. Além dos dois pontos de limite descritos, existe um terceiro ponto pertencente ao exercício de extensão lombar, que é um ponto central entre o limite inferior e o superior. Este ponto central é obtido através de uma simples formulação matemática dentro do *script Cruiser.cs*.

A movimentação do avatar é controlada, em tempo real, pelo OpenPose através dos pontos 1 (pescoço) e 8 (ponto central do quadril) da coluna, presentes na Figura 3.24.

Neste trabalho é definido tempo real como sendo a geração de conteúdo interativo, através de computação gráfica, mais rápido do que a percepção humana.

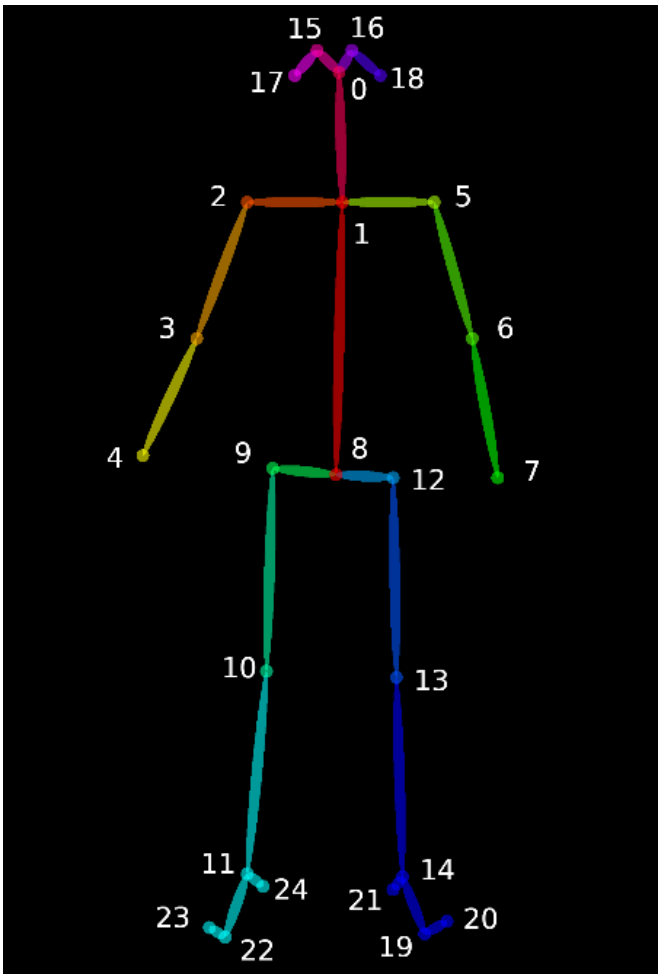

Figura 3.24 – Pontos utilizados pelo OpenPose para detecção das articulações do corpo humano.

Dentro do objeto de jogo referente ao modelo utilizado, tem-se dois objetos filhos. O primeiro é o elemento "*FirePoint*", responsável por fazer surgir os objetos que representam, visualmente, os tiros disparados pela nave (avatar) com intuito de destruir os asteroides que vão surgindo na tela. A Figura 3.25 mostra este elemento na tela de jogo, localizado no ponto de encontro dos três eixos destacados na imagem.

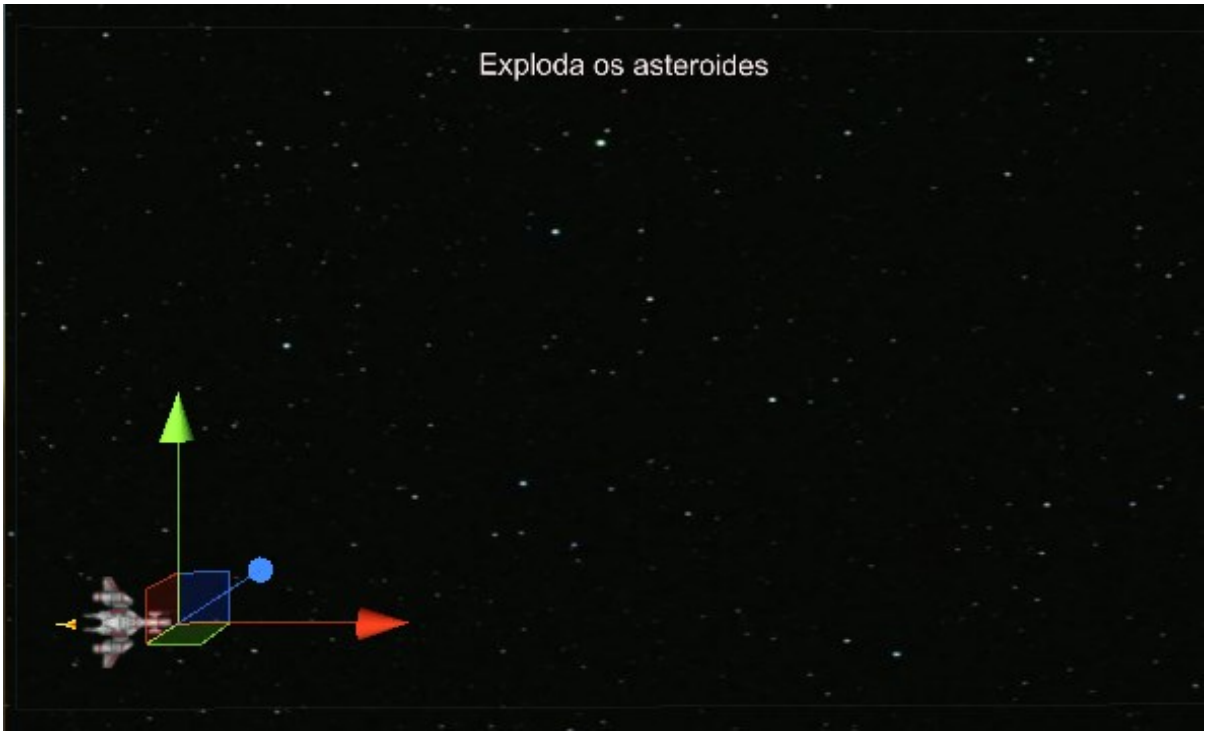

Figura 3.25 – Elemento *FirePoint* e sua localização na tela do jogo “Asteroides”.

Como é mostrado na Figura 3.25, o elemento *FirePont* é um objeto de jogo vazio, posicionado bem a frente da nave que representa o personagem principal e se movimenta junto com ela. Nele está contido apenas o *script spawnbullet.cs* que é responsável por fazer surgir, de maneira periódica, com tempo de início do primeiro “*spawn*” e tempo de *delay* entre os próximos disparos, o *Prefab Bullet* no ponto onde está posicionado o objeto de jogo. O *script spawnbullet.cs* é disponibilizado no Apêndice I.

O *Prefab Bullet* também possui um *script* denominado “*Bullet.cs*”, responsável por controlar a velocidade com que o objeto se movimenta na tela, para dar a impressão visual do tiro sendo disparado em direção aos asteroides que vão surgindo. Este *script* está disponível no Apêndice J.

A Figura 3.26 mostra o *Prefab Bullet*, que foi obtido juntamente com a nave em um dos *assets* disponíveis gratuitamente para download na internet, cujo qual será detalhado no Apêndice B.

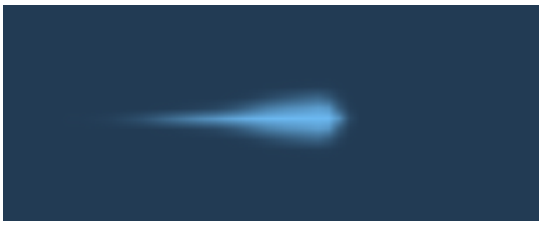

Figura 3.26 – *Prefab Bullet* em detalhe.

O outro elemento filho do objeto *Cruiser*, é o denominado “Propulsão”, que nada mais é que um objeto de jogo posicionado na parte traseira da nave, contendo uma animação formada por *sprites* em sequência, dando a sensação visual de um jato propulsor que movimenta a nave na cena do jogo. A Figura 3.24 (a) mostra o jato propulsor em detalhe.

3.3.5.1.3. Asteroides inimigos

Os asteroides são os objetos de jogo responsáveis por serem os objetivos a serem destruídos pelo jogador. Eles surgem em pontos, e períodos de tempo, específicos e pré-definidos na tela de maneira a fazer com que o paciente faça os movimentos necessários que o exercício de extensão lombar (método McKenzie) exige. A Figura 3.27 mostra um exemplar do asteroide utilizado.

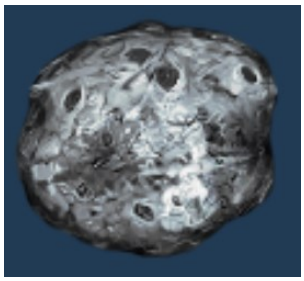

Figura 3.27 – Exemplar de asteroide utilizado no jogo “Asteroides”.

A Figura 3.28 mostra o movimento da coluna controlando a nave para destruir os asteroides que vão surgindo na tela.

Figura 3.28 – Controlando o avatar do jogo e destruindo os asteroides através dos movimentos da coluna.

Existem três objetos de jogo denominados “Asteroid”, “Asteroid1” e “Asteroid2”, posicionado na tela nas mesmas três posições em que a espaçonave se movimenta. Cada um dos objetos de jogo tem um *spawn* de asteroides, em tempos pré-definidos, com o intuito de

levar o paciente a realizar o exercício de maneira correta. Estes objetos de jogo possuem um *script* denominado "*spawnAsteroid.cs*", responsável por fazer surgir os asteroides nos devidos pontos e espaços de tempo. Este *script* está disponível no Apêndice K.

Os asteroides que surgem nestes pontos de *spawn* são *Prefabs*, assim como o tiro disparado pela nave. Estão contidos nestes *Prefabs* uma animação de movimentação dos mesmos, com os nomes *Asteroid, Asteroid2* e *Asteroid3*, e um script denominado "*ast.cs*".

A animação leva ao jogador a ilusão visual de que os asteroides estão indo em direção à nave, porém param de se movimentar em um certo ponto da tela próximo do local em que surgiram, a fim de não forçar movimentos bruscos por parte do paciente devido a um possível "nervosismo" ao observar um asteroide se aproximando muito da nave. Já o *script* "*ast.cs"* é responsável por fazer o controle da pontuação do jogo, onde é acrescido um ponto ao marcador a cada asteroide destruído pelo avatar de espaçonave controlado pelo jogador, além de fazer o controle de destruição dos clones *Bullet*, *Asteroid, Asteroid2* e *Asteroid3*. Este *script* está disponível no Apêndice L.

Os asteroides são destruídos quando os *colliders* do *Prefab Bullet* e do próprio asteroide, presente na tela, tem contato entre si. Tanto o tiro disparado pela nave quanto o asteroide desaparecem da tela e neste mesmo local surge uma animação de explosão denominada "*explosion_0*", cuja qual também é um *Prefab*. Esta animação é feita a partir de vários sprites em sequência reunidas através do elemento "*Animator"* presente na janela "*Inspector"*. Os sprite em sequência podem ser visualizados na Figura 3.29.

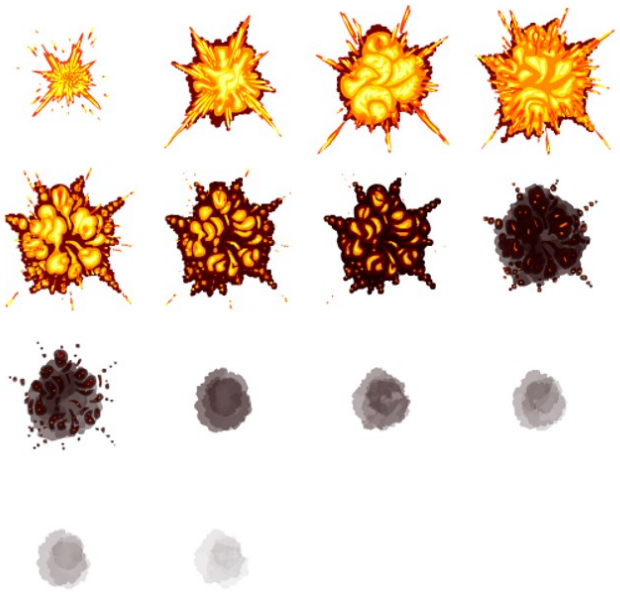

Figura 3.29 – Sprites em sequência que formam a animação *esplosion_0.*

O resultado visual obtido na tela de jogo pode ser observado na Figura 3.30.

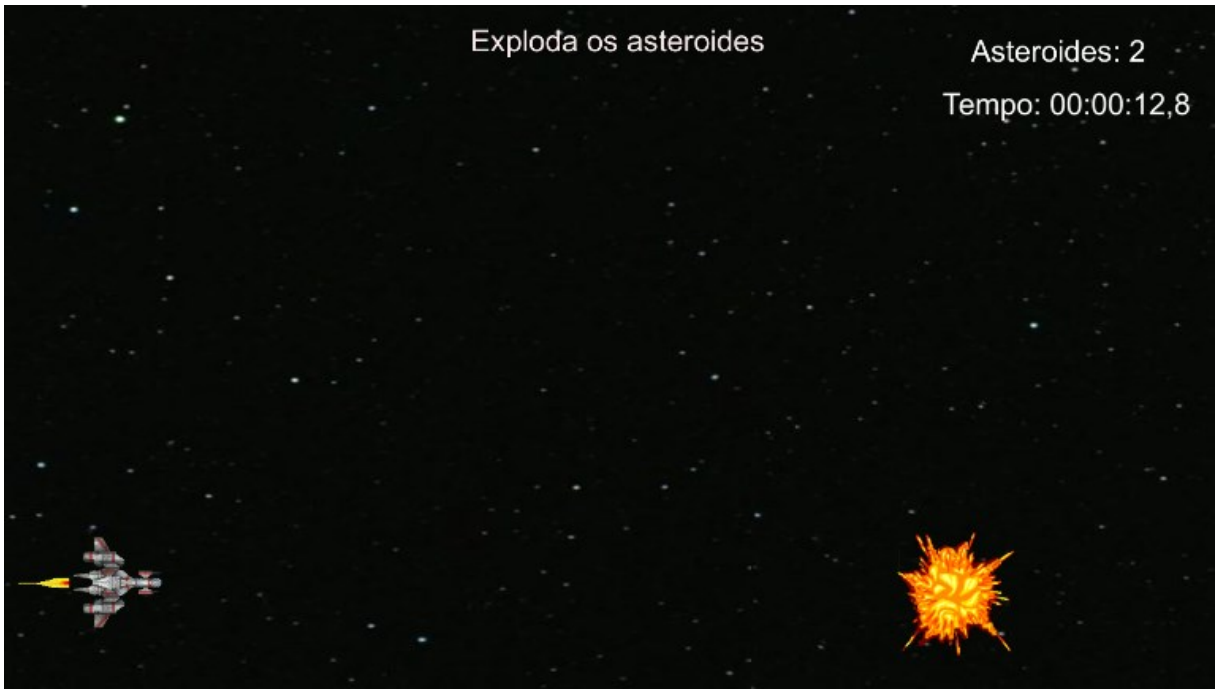

Figura 3.30 - Representação visual da explosão de um dos asteroides.

A contagem de tempo dos jogos é apresentada na tela em um objeto de jogo do tipo "*Text*", posicionado no canto superior direito, denominado "Contador_asteroides". É possível visualizar o contador com o placar de 2 asteroides destruídos na Figura 3.31.

Também é possível se visualizar, nesta Figura 3.31, o contador de tempo de execução do jogo, logo abaixo do contador de pontos. Este contador de tempo é um objeto de jogo, também do tipo "*Text*". O sistema de tempo é utilizado com base no sistema de temporização do próprio Unity 3D, da classe *Time*. Ele obtém, a cada frame, o tempo decorrido desde o carregamento da cena atual em segundos.

O script "Tempo.cs", disponível no Apêndice M, faz com que este tempo obtido a cada frame seja convertido em uma *string* no formato ##:##:##,# para que fique mais compreensível para o usuário o tempo decorrido durante uma partida.

Os valores de pontuação e de tempo são salvos no mesmo arquivo de relatório gerado na cena inserção de paciente, a partir do *script* "Salvararquivo.cs", no apêndice E.

Este *script* obtém os valores no painel Canvas e, quando acionado o menu de pause durante a execução do jogo, ao se clicar no botão "salvar" é acionado e executado. Após o clique, ele abre o arquivo existente a partir do nome do arquivo de relatório salvo na cena inserção de paciente, passado por meio da classe *PlayerPrefs*, que consegue transmitir

informações entre cenas do jogo. Além da pontuação e do tempo, ao se clicar no botão “salvar” do menu de pause, o nome do jogo que o paciente está jogando no momento também é salvo no arquivo gerado na cena de inserção do paciente.

Sendo assim, as três últimas linhas da Figura 3.16 são adicionadas ao arquivo quando o usuário clica no botão “salvar” do menu de pause do jogo.

3.3.5.2. Jogo “Dia de Parque”

Nesta seção será apresentada a interface principal do jogo “Dia de Parque”, bem como seus elementos e funcionamento.

3.3.5.2.1. Interface principal do jogo “Dia de Parque”

Nesta cena é apresentado o segundo jogo em execução. Aqui, assim como nos demais jogos, o paciente irá interagir com o personagem principal do jogo através dos movimentos capturados pela câmera e processados pelo *OpenPose.*

A Figura 3.31 mostra a tela padrão desta cena. Nela está o personagem controlável pelo jogador através do *OpenPose*, via câmera, cujo qual é representado por uma criança que incentiva os animais a chegarem nos seus donos. O cenário representa um parque arborizado com animais que interagem com o usuário de maneira a passar a sensação de calma e tranquilidade que um parque trás.

Figura 3.31 – Tela principal do jogo “Dia de Parque”.

Na imagem é possível se observar os três animais que devem ser incentivados a chegarem em seus respectivos donos. Na parte esquerda estão o cachorro, o coelho e o gato, cujos quais devem chegar em seus donos na respectiva ordem na parte direita do parque.

Também é possível se observar na Figura 3.31, na parte superior central, uma frase explicando o que o jogador deve fazer, funcionando como o tutorial do jogo. Já na parte direita está o placar de pontuação do jogo. Os demais animais da cena tem o objetivo de inserir o usuário mais ainda no contexto de parque alguns até com o cunho cómico no intuito de entreter o paciente.

3.3.5.2.2. Avatar do jogo "Dia de Parque"

O avatar é o personagem principal do jogo tem a função de simular para o ambiente virtual os movimentos da coluna que o usuário paciente faz em frente a câmera. Este personagem é um modelo pré-fabricado (*Prefab*) disponibilizado gratuitamente na internet.

Ele representa uma criança que incentiva os "*pets*" a correrem até seus respectivos donos e responde ao movimento de dois pontos específicos da coluna, sendo um deles, localizado no pescoço do paciente, e o outro na parte central do quadril. Os dois exercícios que o jogo propõe ao paciente são o de flexão lombar (posição ereta) e o de extensão lombar (método McKenzie – posição ereta). O modelo utilizado pode ser visualizado na Figura 3.32.

Figura 3.32 – Modelo utilizado para o avatar do jogo "Dia de Parque".

Este modelo é constituído de uma animação feita a partir de vários *sprites*. Estes *sprites* eram originalmente um *gif* que foi convertido. A Figura 3.33 mostra os sprites após a conversão.

Figura 3.33 - *Sprites* utilizados na montagem da animação do avatar.

O avatar acompanha os movimentos feitos pelo usuário em frente à câmera, referentes aos exercícios de flexão e extensão lombar. Estes movimentos respeitam os limites de angulação que o paciente consegue realizar com a coluna, cujos quais foram delimitados na cena "Calibração", anteriormente citada. Para o controle destes movimentos no avatar é utilizado o script "*PlayerCrianca.cs*", cujo qual está disponível no Apêndice N.

Este *script* faz com que o personagem se movimente entre três posições específicas, a fim de cumprir o proposto pelo exercício fisioterápico. As posições são obtidas através da coluna do paciente em dois pontos pré-estabelecidos, marcadas nas quatro variáveis tipo "*PlayerPrefs*" na tela de calibração, citadas no item *3.3.5.1.2.*, para os respectivos exercícios do jogo "Dia de Parque". Estes pontos são tidos como o limite superior e limite inferior de movimentação da coluna.

Tais pontos são interpretados através do script *PlayerCrianca.cs* e são acrescidos a estes valores uma margem de posicionamento, tanto para mais, quanto para menos, em que, ao movimentar a coluna, o paciente tenha visualmente a confirmação que está se posicionado corretamente durante o exercício. O ponto central é o ponte de união dos dois exercícios e também o ponte de início de ambos.

Assim como descrito no Item *3.3.5.1.2.,* a movimentação do avatar é controlada em tempo real, pelo OpenPose, através dos pontos 1 (pescoço) e 8 (ponto central do quadril) da coluna, presentes na Figura 3.24.

A Figura 3.34 mostra o movimento da coluna controlando o avatar que por sua vez aciona a movimentação dos animais [personagens secundários] no jogo.

Figura 3.34 – Movimentando o avatar do jogo ”Dia de Parque” através da coluna.

3.3.5.2.3. Personagens secundários do jogo “Dia de Parque”

O objetivo do jogo é fazer com que os pets cheguem até seus respectivos donos, sendo assim a criança, avatar do jogo, os incentiva a correr na horizontal.

Cada pet é um personagem secundário e tem sua própria animação individual, todas vindas de *sprites* convertidos de *gif's* e posteriormente animados no Unity assim como o avatar do jogo.

Analisando verticalmente a Figura 3.32, no sentido de cima para baixo, tem-se um cachorro, um coelho e um gato, todos em posições verticais fixas e se movimentando linearmente na horizontal, partindo da esquerda da tela até a parte direita, em encontro aos seus respectivos donos.

A Figura 3.35 mostra o primeiro personagem secundário representado pelo cachorro.

Figura 3.35 - Primeiro personagem secundário representado por um cachorro.

Já a Figura 3.36 mostra os sprites após a conversão do gif original do cachorro.

Figura 3.36 - Sprites utilizados para a animação do primeiro personagem secundário, o cachorro.

O personagem “cachorro” tem como ponto de chegada no jogo, sua “dona”, representada por uma personagem feminina na parte direita do parque. Esta personagem também é uma animação oriunda de sprites convertidos de um *gif*. A Figura 3.37 mostra esta personagem.

Figura 3.37 – Personagem feminina “dona” do personagem cachorro.

Os sprites utilizados na montagem da animação são representados na Figura 3.38.

Figura 3.38 - Sprites utilizados para a animação da “dona” do personagem secundário cachorro.

A Figura 3.39 mostra o segundo personagem secundário representado pelo coelho.

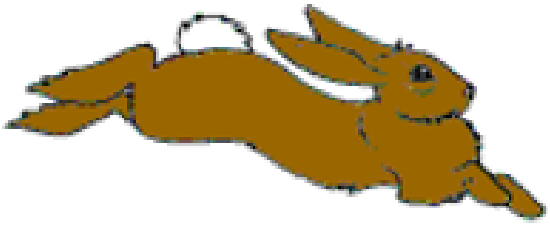

Figura 3.39 – Segundo personagem secundário representado por um coelho.

Já a Figura 3.40 mostra os sprites após a conversão do gif original do coelho.

Figura 3.40 - Sprites utilizados para a animação do segundo personagem secundário, o coelho.

O personagem “coelho” também tem como ponto de chegada no jogo sua “dona” representada por uma personagem feminina na parte direita do parque. A Figura 3.41 mostra esta personagem.

Figura 3.41 – Personagem feminina “dona” do personagem coelho.

Os sprites utilizados na montagem da animação são representados na Figura 3.42.

Figura 3.42 - Sprites utilizados para a animação da “dona” do personagem secundário coelho.

A Figura 3.43 mostra o terceiro personagem secundário representado pelo gato.

Figura 3.43 – Terceiro personagem secundário representado por um gato.

Já a Figura 3.44 mostra os sprites após a conversão do gif original do gato.

Figura 3.44 - Sprites utilizados para a animação do terceiro personagem secundário, o gato.

Assim como os personagens “cachorro” e “coelho”, o personagem “gato” também tem como ponto de chegada no jogo, sua “dona” representada por uma personagem feminina na parte direita do parque. A Figura 3.45 mostra esta personagem.

Figura 3.45 - Personagem feminina “dona” do personagem *gato.*

Os sprites utilizados na montagem da animação são representados na Figura 3.46.

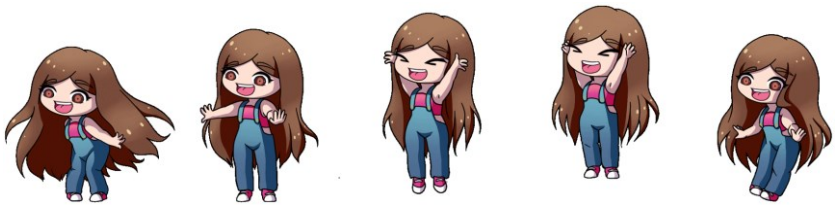

Figura 3.46 - Sprites utilizados para a animação da “dona” do personagem secundário gato.

3.3.5.2.4. Demais componentes do cenário

Na cena também existem os componentes de cunho cómico e os que representam a "torcida animal" do jogo. Estes componentes tem o intuito de entreter e manter o interesse do jogador fazendo com que o mesmo mantenha o exercício durante mais tempo.

Assim como o avatar e os personagens principais estes objetos de jogo são animações feitas com sprites convertidos de *gif's*.

A Figura 3.47 mostra os componentes da "torcida animal".

Figura 3.47 – Personagens integrantes da "torcida animal".

Na Figura 3.48 são apresentados os sprites após a conversão do gif original.

Figura 3.48 - Sprites utilizados para a animação da "torcida animal".

A seguir serão apresentados os componentes que tem a função de entreter o jogador comicamente.

A Figura 3.49 mostra o componente "gato sonolento" uma animação que representa um gato dormindo.

Figura 3.49 – Personagem de cunho cómico, o "gato sonolento".

Na Figura 3.50 são apresentados os sprites após a conversão do gif original.

Figura 3.50 – Sprites utilizados para a animação do componente “gato sonolento”.

A Figura 3.51 mostra o “gato brincalhão” o componente que representa um gato brincando de rolar no chão.

Figura 3.51 – Personagem de cunho cómico, o “gato brincalhão”.

Na Figura 3.52 são apresentados os sprites após a conversão do gif original.

Figura 3.52 - Sprites utilizados para a animação do componente “gato brincalhão”.

A Figura 3.53 mostra o “cachorro do balão” o componente que representa um cachorro amarrado em um balão de festa e que fica planando sob as arvores do parque durante a execução do jogo. Esta é uma animação feita com apenas dois sprites que ficam alternando as posições de altura em que o cachorro plana.

Figura 3.53 - Personagem de cunho cómico, o “cachorro do balão”.

Na Figura 3.54 são apresentados os sprites após a conversão do gif original.

Figura 3.54 - Sprites utilizados para a animação do componente “cachorro do balão”.

Por fim, a Figura 3.55 mostra borboletas que representam a serenidade e tranquilidade, que espera levar o jogador a ter o sentimento de imersão na natureza.

Figura 3.55 - Componente de jogo “borboletas”.

Na Figura 3.56 são apresentados os sprites após a conversão do gif original.

Figura 3.56 - Sprites utilizados para a animação do componente “borboletas”.

3.3.5.2.5. Contagem de tempo e pontuação.

É possível observar no canto superior direito da Figura 3.32 o contador de pontos. Os pontos são contados a partir da chegado dos pets, inicialmente na parte esquerda do parque, aos seus donos, na parte direita. Cada pet que chega ao seu respectivo dono soma 1 ponto ao contador. O *script* responsável por fazer o controle da pontuação dentro do jogo é o “PlayerCrianca.cs”, contido no Apêndice N.

O contador de tempo de execução do jogo aparece na tela com o início do mesmo logo abaixo do contador de pontos. Ele é um objeto de jogo também do tipo “*Text*”. O sistema de

tempo 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.

Assim como nos jogos “Asteroides”, é utilizado o script “Tempo.cs”, disponível no Apêndice M, o qual faz com que este tempo obtido a cada frame seja convertido em uma *string* no formato ##:##:##,#, no intuito de ficar mais compreensível para o usuário o tempo decorrido durante uma partida.

Também como em “Asteroides” os valores de pontuação e de tempo são salvos no mesmo arquivo de relatório gerado na cena inserção de paciente, a partir do *script* “Salvararquivo.cs”, no Apêndice E.

Este *script* obtém os valores no painel Canvas e, quando acionado o menu de pause durante a execução do jogo, ao se clicar no botão “salvar” é acionado e executado. Após o clique, ele abre o arquivo existente a partir do nome do arquivo de relatório salvo na cena inserção de paciente, passado por meio da classe *PlayerPrefs*, que consegue transmitir informações entre cenas do jogo. Além da pontuação e do tempo, ao se clicar no botão “salvar” do menu de pause, o nome do jogo que o paciente está jogando no momento também é salvo no arquivo gerado na cena de inserção do paciente.

Sendo assim, as três últimas linhas da Figura 3.16 são adicionadas ao arquivo quando o usuário clica no botão “salvar” do menu de pause do jogo.

3.3.5.3. Jogo “Pesca”

Nesta seção será apresentada a interface principal do jogo “Pesca”, bem como seus elementos e funcionamento.

3.3.5.3.1. Interface principal do jogo “Pesca”

Nesta cena é apresentado o terceiro jogo em execução. Assim como nos demais jogos o paciente irá interagir com o personagem principal do jogo através dos movimentos capturados pela câmera e processados pelo *OpenPose*.

A Figura 3.57 mostra a tela padrão desta cena. Nela estão os personagens controláveis pelo jogador através do *OpenPose*, via câmera, os quais são um pescador e sua plataforma elevadora, uma arraia voadora, em um cenário de pescaria representado por uma paisagem de natureza, com cachoeiras, um rio, árvores ao fundo e muitos peixes pulando na água.

Figura 3.57 – Tela principal do jogo “Pesca”.

Este jogo tem o intuito de induzir o jogador a realizar os movimentos necessários para fazer de maneira correta o exercício de flexão do quadril (exercícios de Williams) na posição deitado e com a parte traseira do corpo em direção ao chão. Nele o jogador tem o objetivo de pescar o maior número de peixes que conseguir, peixes esses que estão a todo momento pulando da água na tela do jogo.

Na Figura 3.57 pode ser observado, no centro superior da tela, uma frase explicando o que o jogador deve fazer, funcionando como o tutorial do jogo.

Também na Figura 3.57 é possível se observar, no canto superior direito, o tempo de execução do jogo e , no canto inferior direito, a pontuação que o jogador tem até o momento.

Com uma proposta diferente do primeiro jogo e seguindo o método de construção do segundo, o “Pesca” é todo feito com gifs, que foram transformados em sprites e utilizados para montagem de animações dentro de jogo. As animações finais vão desde o movimento da água e dos peixes pulando dela, até o movimento da arraia e do pescador.

A Figura 3.58 mostra o modelo utilizado para ser o plano de fundo do jogo.

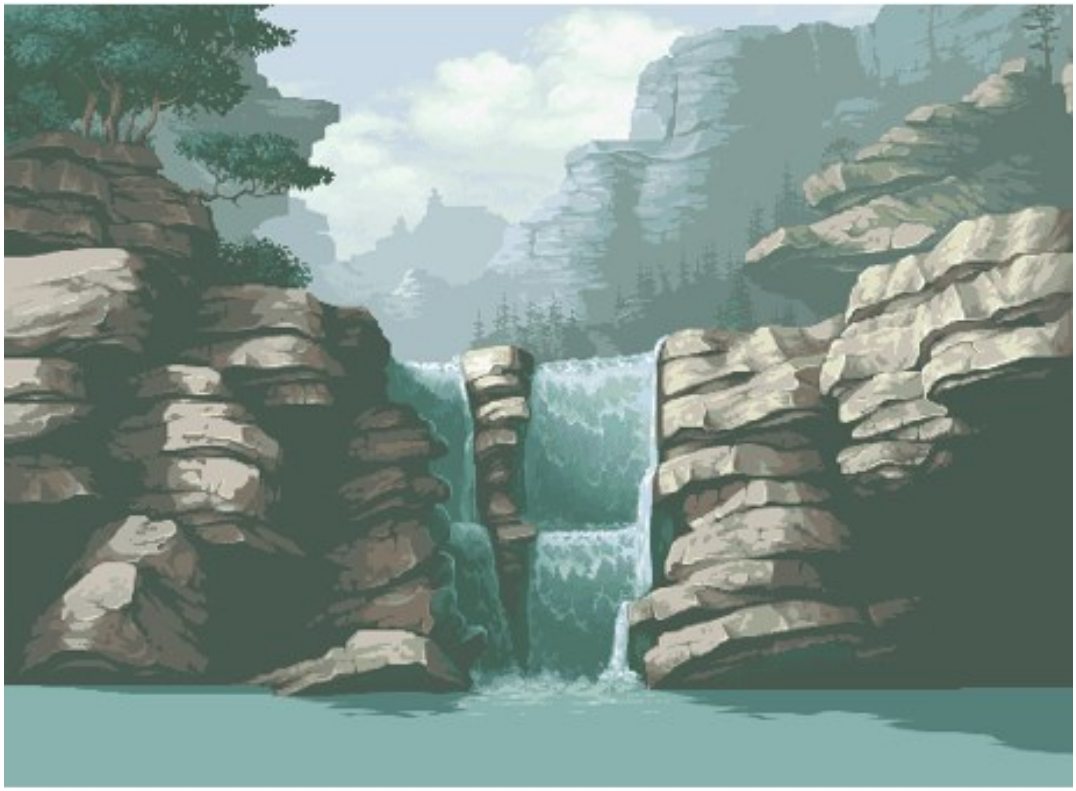

Figura 3.58- Plano de fundo do jogo "Pesca".

Já na figura 3.59 são apresentados os *sprites* utilizados para fazer a animação deste plano de fundo, dando a impressão ao jogador de movimentação da água na cachoeira.

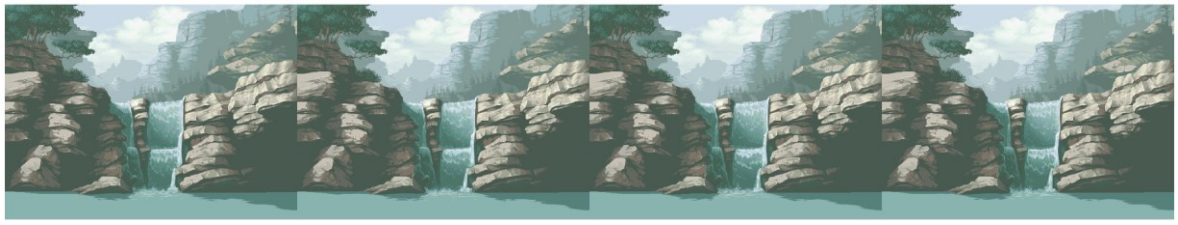

Figura 3.59 – *Sprites* utilizados na montagem da animação do plano de fundo – jogo “Pesca”.

3.3.5.3.2. Avatar do jogo “Pesca”

Assim como explicado na seção *3.3.5.1.2. - Avatar do jogo “Asteroides”*, o avatar é o personagem principal do jogo e tem a função de simular para o ambiente virtual os movimentos da coluna do usuário paciente. Como nos demais jogos, o avatar do jogo *Pesca* é um *Prefab* e seus *sprites* foram obtidos gratuitamente na internet.

Este avatar representa o já mencionado pescador, que por mais que ele esteja sempre acompanhado de sua arraia voadora, é apenas ele o responsável por simular os movimentos de pescaria e por fornecer pontos ao jogador por pescar os peixes. Tanto o pescador quanto a arraia, respondem ao movimento de dois pontos específicos da coluna, sendo um deles, localizado no

pescoço do paciente, responsável por efetuar os movimentos do personagem de fato. O outro ponto, localizado na parte central do quadril do paciente, é utilizado como referência para verificar se o mesmo está corretamente posicionado para realizar o exercício em questão.

O avatar é chamado à tela através do *script* "*spawnpescador.cs"*, inserido em um objeto de jogo vazio posicionado na tela. Tal *script* também é responsável por fazer o controle de posição do avatar, através da análise dos movimentos do jogador, capturados via câmera e analisados pelo *OpenPose*. O *script spawnpescador.cs* pode ser visualizado no Apêndice P.

Este *script* verifica a cada frame a posição atual do jogador e, caso ela esteja dentro dos limites de uma das três posições pré-estabelecidas, ele faz um *spawn* do avatar naquela posição, caso o jogador permaneça nesta posição durante o tempo de execução da animação (que é o pescador fazendo um movimento de pescar um peixe), um peixe é pescado e é acrescentado um ponto ao placar de peixes. A Figura 3.60 mostra o movimento da coluna controlando o avatar que por sua vez realiza a ação de pescar peixes.

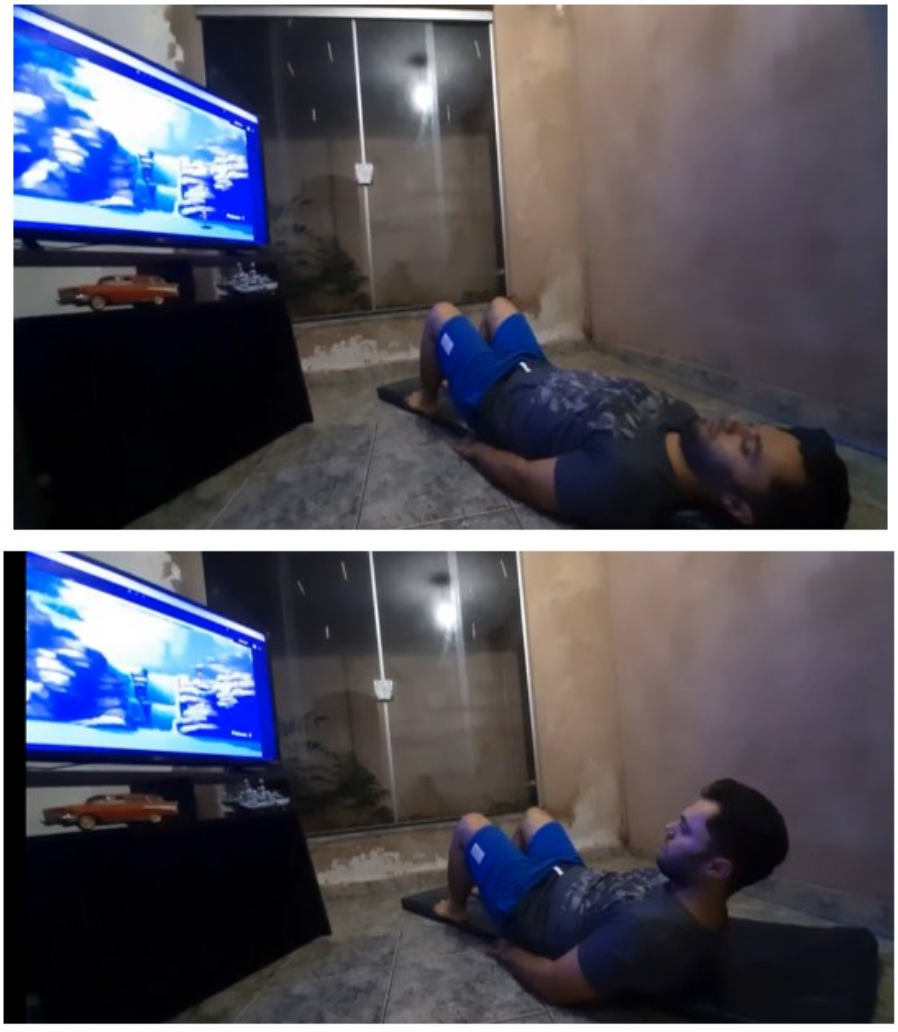

Figura 3.60 - Movimentando o avatar do jogo "Pesca" através da coluna.

O modelo utilizado é feito em pixel art (arte pixel), que é uma forma de arte digital na qual as imagens são criadas ou editadas tendo como elemento básico os pixels. Tal modelo pode ser visualizado na Figura 3.61.

Figura 3.61 – Modelo utilizado para o avatar do jogo "Pesca".

A Figura 3.62 mostra os *sprites* obtidos após a conversão do gif original.

Figura 3.62– *Sprites* resultantes da conversão do gif original do avatar pescador.

3.3.5.3.3. Plataforma elevadora "arraia"

A arraia, assim como os demais componentes do jogo, é um objeto de jogo composto por uma animação feita a partir de *sprites* convertidos de um *gif*. Ela funciona de maneira lúdica passando ao usuário a impressão de elevar o avatar pescador como se fosse uma plataforma.

O *script* responsável por fazer a movimentação da arraia é o "*arraia.cs"*, disponível no Apêndice Q, e, assim como no avatar do pescador, este *script* capta os movimentos do jogador através da câmera e os traduz com o *OpenPose* para movimentar a arraia nas três posições pré-definidas que o pescador possui, sendo posicionada sempre abaixo dos pés do mesmo. Os pontos no corpo do jogador, responsáveis pela movimentação da arraia, são os mesmos que o avatar do pescador analisa em seu *script* de controle.

A Figura 3.63 mostra o modelo utilizado para a montagem da arria.

Figura 3.63 – Modelo utilizado para a criação da arraia no jogo "Pesca".

Na Figura 3.64 é possível se observar os *sprites* obtidos após a conversão do *gif* original.

Figura 3.64 – *Sprites* resultantes da conversão do gif original da arraia.

3.3.5.3.4. Peixes

Para aumentar a imersão do jogador e transmitir uma cena de pescaria empolgante e chamativa, foram implementados na cena do jogo várias animações de peixes pulando a água. Existem três diferentes tipos de peixes e todas as animações, assim como os demais componentes de jogo foram feitas a partir de *sprites* convertidos de *gif's*.

Foram feitas cópias dos três peixes pré-fabricados e colocadas em locais estratégicos na cena. Em todos estes objetos de jogo vazios foram adicionados o *script* "*peixe.cs"*, o qual é responsável pelo *spawn* dos peixes em tempos pré-definidos para passar uma experiência de pesca mais atrativa ao jogador no intuito de motivar o mesmo a praticar o exercício fisioterápico por mais tempo. O *script* "*peixe.cs"* pode ser visualizado no Apêndice R.

A primeira animação a ser citada foi feita a partir do modelo *Prefab* "peixao_0". A Figura 3.65 mostra este modelo.

Figura 3.65 – Modelo utilizado para o peixao_0.

Os *sprites* obtidos após a conversão do *gif* original podem ser observados na figura 3.66.

Figura 3.66 – *Sprites* obtidos após a conversão do gif original do peixe "peixão_0".

Já a segunda animação foi feita a partir do modelo *Prefab* "peixe2_0". A Figura 3.67 mostra este modelo.

Figura 3.67 – Modelo utilizado para o "peixe2_0".

Os *sprites* obtidos após a conversão do *gif* original podem ser observados na Figura 3.68.

Figura 3.68 - *Sprites* obtidos após a conversão do gif original do peixe “peixe2_0”.

Por fim, a terceira animação foi feita a partir do modelo *Prefab* “peixinho_2”. Este modelo é praticamente idêntico ao modelo peixão_0, porém seu tamanho em cena é menor. A Figura 3.69 mostra este modelo.

Figura 3.69 - Modelo utilizado para o peixinho_2.

Os *sprites* obtidos após a conversão do *gif* original podem ser observados na Figura 3.70.

Figura 3.70 - *Sprites* obtidos após a conversão do gif original do peixe “peixinho_2”.

3.3.5.3.4. Contagem de tempo e pontuação

Assim como no jogo “Asteroides” e no jogo “Dia de Parque”, o contador de tempo de execução do jogo, no canto superior direito, é um objeto de jogo, também do tipo “*Text*”. O sistema de tempo 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.

Também é utilizado o script “Tempo.cs”, disponível no Apêndice M, cujo qual faz com que este tempo obtido a cada frame seja convertido em uma *string* no formato ##:##:##,#.

Assim como dito em *3.3.5.3.2.*, caso o jogador permaneça em uma das três posições pré-estabelecidas durante o tempo de execução da animação (que é o pescador fazendo um movimento de pescar um peixe), um peixe é pescado e é acrescentado um ponto ao placar de peixes.

Os dados salvos são adicionados às três últimas linhas do arquivo da Figura 3.16 quando o usuário clica no botão “salvar” do menu de pause do jogo.

3.3.5.4. Jogo “Corrida Infinita”

Nesta seção será apresentada a interface principal do jogo “Corrida Infinita”, bem como seus elementos e funcionamento.

3.3.5.4.1. Interface principal do jogo “Corrida Infinita”

Nesta cena é apresentado o quarto jogo em execução. Assim como nos demais jogos, o paciente irá interagir com o personagem principal do jogo através dos movimentos capturados pela câmera e processados pelo *OpenPose*.

A Figura 3.71 mostra a tela padrão desta cena. Nela está o personagem controlável pelo jogador através do *OpenPose*, via câmera, o qual é um robô, em um cenário composto por uma ponte infinita em mar aberto.

Figura 3.71 – Tela principal do jogo “Corrida Infinita”.

Este jogo tem o intuito de induzir o jogador a realizar os movimentos necessários para fazer de maneira correta os exercícios de flexão lateral esquerda e flexão lateral direita, na posição ereta. Nele o jogador tem o objetivo de coletar o maior número de moedas possível.

Pode ser observado no centro superior da tela, uma frase explicando o que o jogador deve fazer funcionando como o tutorial do jogo.

Também é possível se observar, no canto superior esquerdo, a distância percorrida em metros e, logo abaixo, a pontuação que o jogador tem até o momento.

A proposta deste jogo é fazer com que o jogador mantenha a posição exigida pelos exercícios propostos, durante o tempo necessário, e, para isso, as moedas vão surgindo na tela em três posições, pré-definidas, alternadamente.

A Figura 3.72 mostra o movimento da coluna controlando o avatar se movimenta na pista coletando as moedas.

Figura 3.72 - Movimentando o avatar do jogo ”Corrida Infinita” através da coluna.

3.3.5.4.2. Avatar do jogo "Corrida Infinita"

Assim como já dito nos demais jogos, o avatar é o personagem principal do jogo, tem a função de simular para o ambiente virtual os movimentos da coluna do usuário paciente, cujo qual estará fazendo os exercícios fisioterápicos em frente a câmera. Este personagem é um modelo pré-fabricado (*Prefab*) disponibilizado gratuitamente na internet para *download* por meio da janela "*Asset Store*".

Ele representa um robô com articulações e responde ao movimento de dois pontos específicos da coluna, sendo um deles, localizado no pescoço do paciente, e o outro ponto localizado na parte central do quadril. O modelo utilizado pode ser visualizado na Figura 3.73.

Figura 3.73 – Modelo utilizado para o avatar do jogo "Corrida Infinita".

Este modelo tem uma animação que, ao se executar o jogo, simula o movimento de corrida do robô. O avatar possui o componente "*Character Controller*" que é usado principalmente para o controle de terceira pessoa que não faz o uso da física Rigidbody (*Unity*, 2021).

Também está inserido no avatar o *script "Player.cs"* responsável por controlar os movimentos horizontais do robô em três posições pré-estabelecidas, a partir dos movimentos do jogador, os quais são captados pela câmera e analisados pelo *OpenPose*. A velocidade do avatar é pré-estabelecida dentro deste *script*.

Este *script* também é responsável por fazer o controle e contagem da distância percorrida pelo personagem, que nada mais é que a conversão do tempo de execução do jogo em uma distância proporcional em metros, dando a impressão ao jogador de que está em uma

corrida, aumentando a imersão dele no jogo. Além disso, o *script Player.cs* também faz o controle da contagem de moedas coletadas pelo jogador à medida que faz o percurso na ponte.

O *script "Player.cs"* está disponível no Apêndice S.

3.3.5.4.3. Plataformas em formato de Ponte

Para o jogo "Corrida Infinita" foram feitas dez plataformas pré-fabricadas, os *Prefabs*, e ao decorrer da execução do jogo, as plataformas são enfileiradas em sequência, formando uma extensa ponte por onde o avatar corre. Ao final de cada plataforma existe um objeto de jogo vazio chamado "*Finalpoint"*, responsável por ser o marco de reciclagem de cada plataforma.

Toda vez que o avatar passa pelo objeto *Finalpoint*, são contadas 5 unidades de distância e então a plataforma é reciclada, passando para a posição final da ponte. Este ciclo se repete indefinidamente.

A reciclagem das plataformas foi pensada de maneira a otimizar o custo computacional durante a execução do jogo, tornando-o mais leve e rodando de maneira mais uniforme.

Todo o controle e spawn de plataformas é feito através do script "*SpawnPlatform.cs"*, disponibilizado no Apêndice T.

À medida que o avatar percorre as plataformas moedas vão aparecendo no caminho e o jogador deve coletá-las ao controlar os movimentos do avatar. Estas moedas são Prefabs inseridos diretamente nas plataformas e continuam surgindo com a reciclagem das plataformas.

A Figura 3.74 mostra uma dessas moedas pré-fabricadas.

Figura 3.74 – Prefab da moeda do jogo "Corrida Infinita".

O controle de *spawn* das moedas e também do objeto de jogo *Finalpoint* é feito através do script "*Platform.cs"*, cujo qual está inserido em todas as 10 plataformas.

A seguir serão apresentados os *Prefabs* das plataformas nas Figuras 3.75 a 3.84.

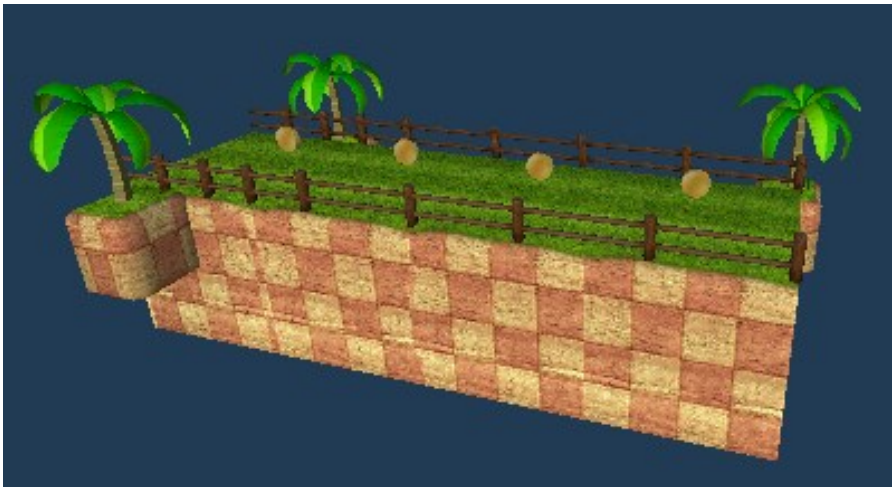

Figura 3.75 – Prefab da Plataforma 1.

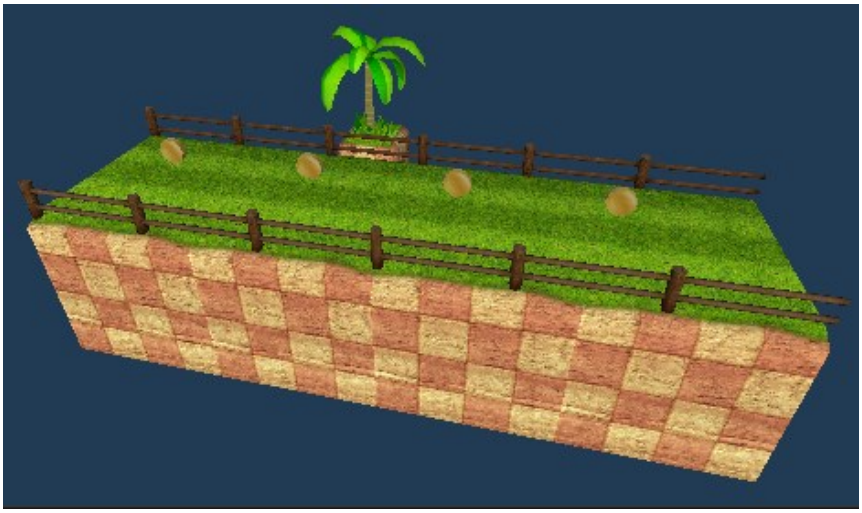

Figura 3.76 - Prefab da Plataforma 2.

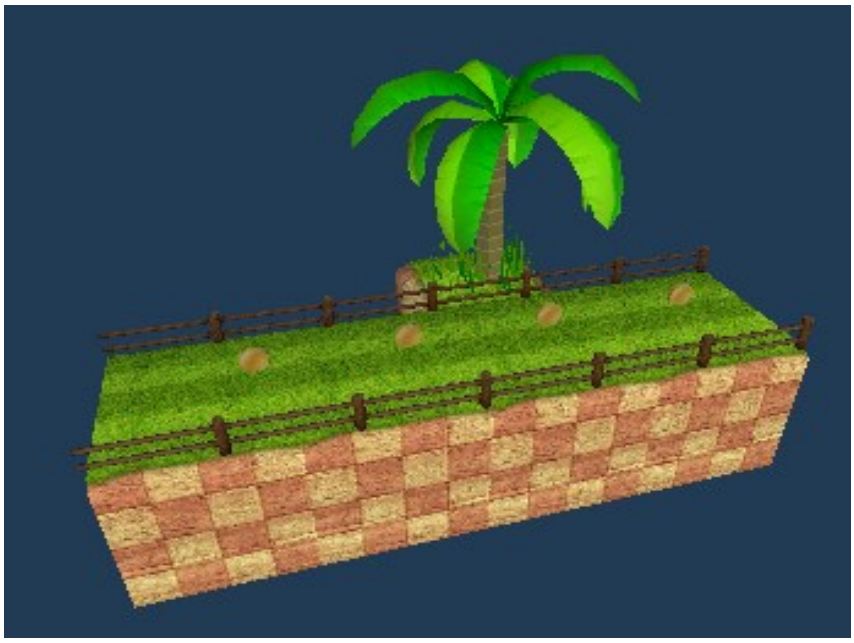

Figura 3.77 - Prefab da Plataforma 3.

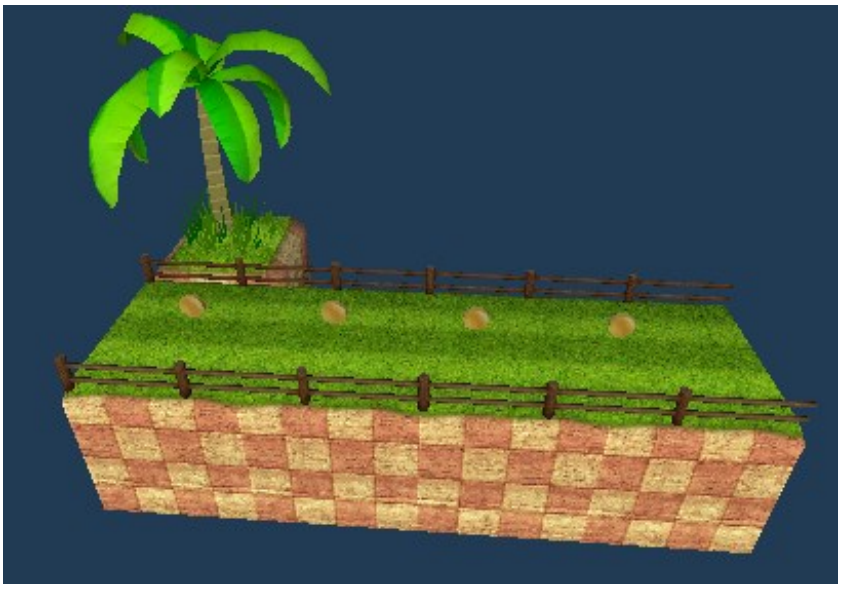

Figura 3.78 - Prefab da Plataforma 4.

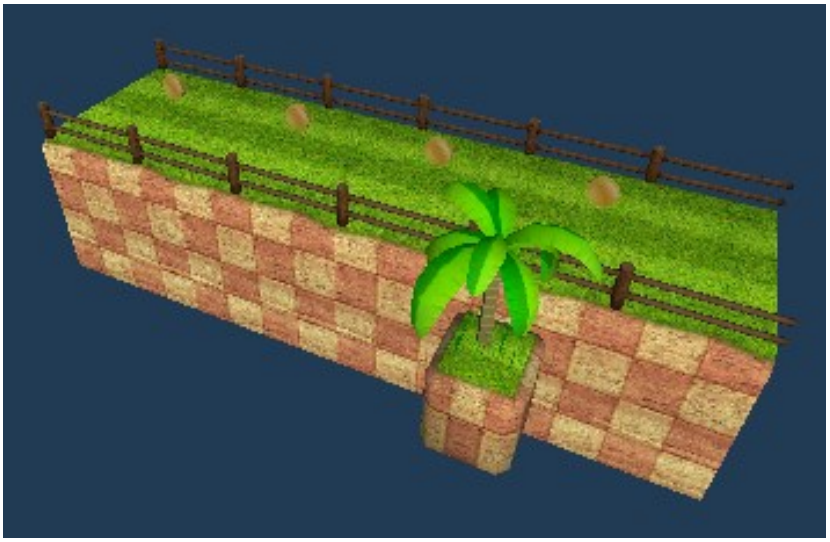

Figura 3.79 - Prefab da Plataforma 5.

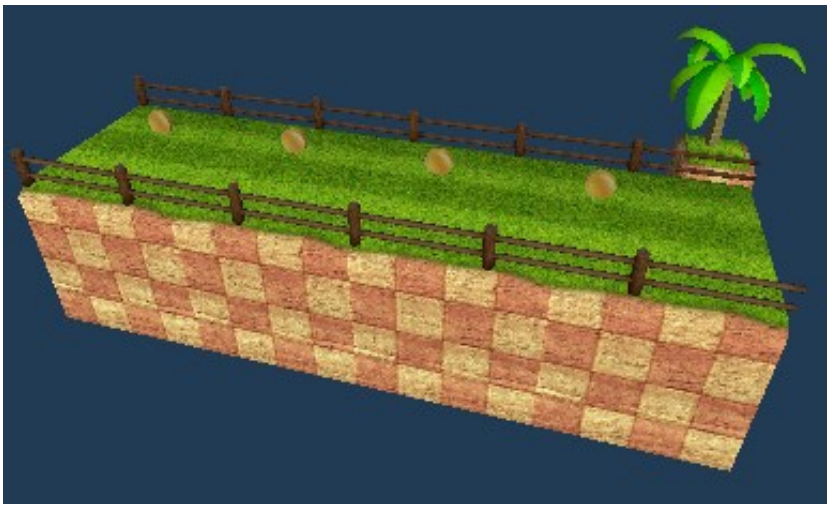

Figura 3.80 - Prefab da Plataforma 6.

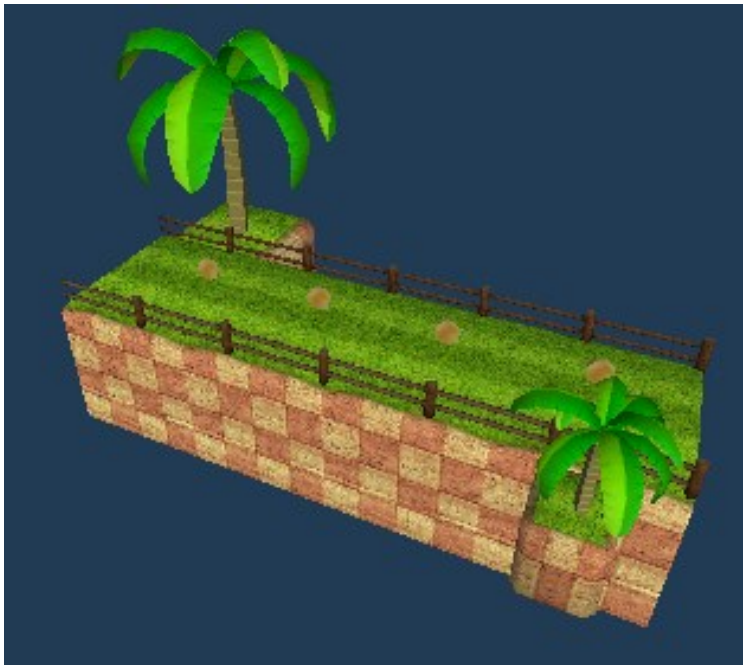

Figura 3.81 - Prefab da Plataforma 7.

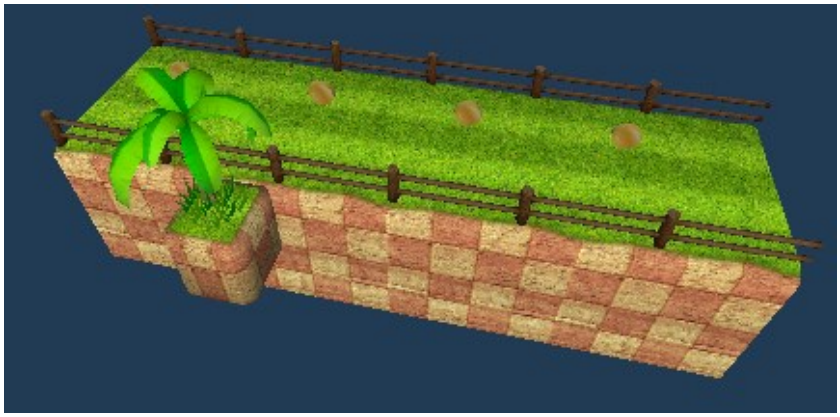

Figura 3.82 - Prefab da Plataforma 8.

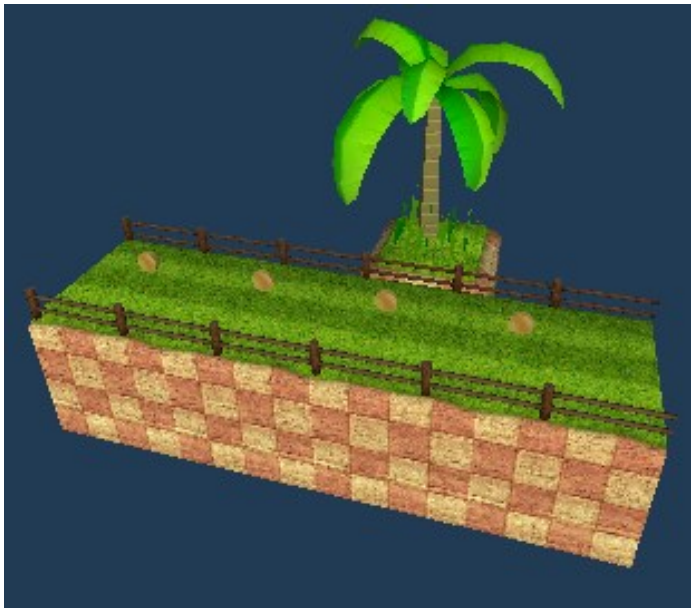

Figura 3.83 - Prefab da Plataforma 9.

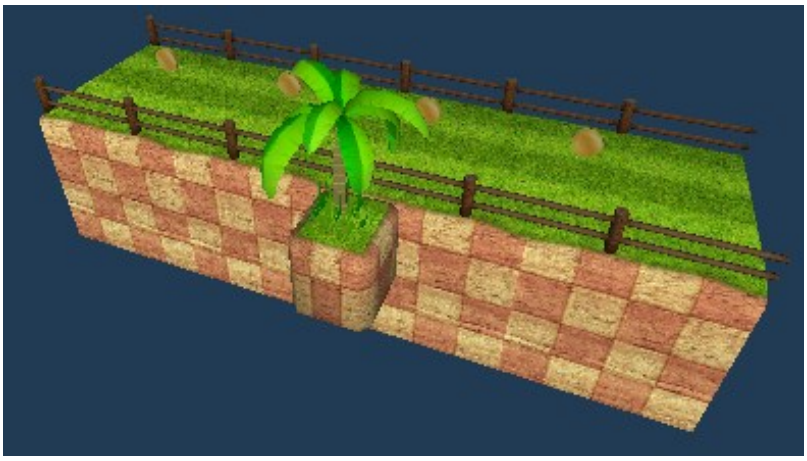

Figura 3.84 - Prefab da Plataforma 10.

O *Prefab* simbolizando o mar também é reciclado assim como as plataformas. No jogo existem dois *Prefabs* iguais em sequência, quando o avatar do robô passa pelo objeto de jogo no final deles, após 5 unidades de medida aquele *Prefab* é reciclado e passa a ficar na posição final da sequência. A Figura 3.85 mostra este *Prefab*.

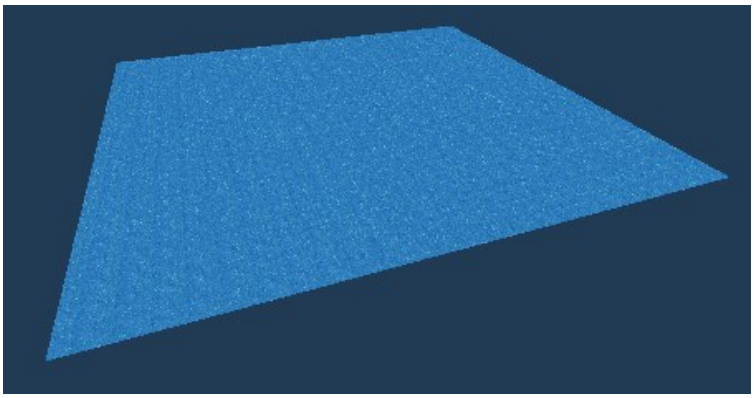

Figura 3.85 – Prefab que simboliza o mar dentro do jogo “Corrida Infinita”.

A montagem completa das plataformas em sequência, bem como os *Prefabs* do mar, podem ser observada na Figura 3.86.

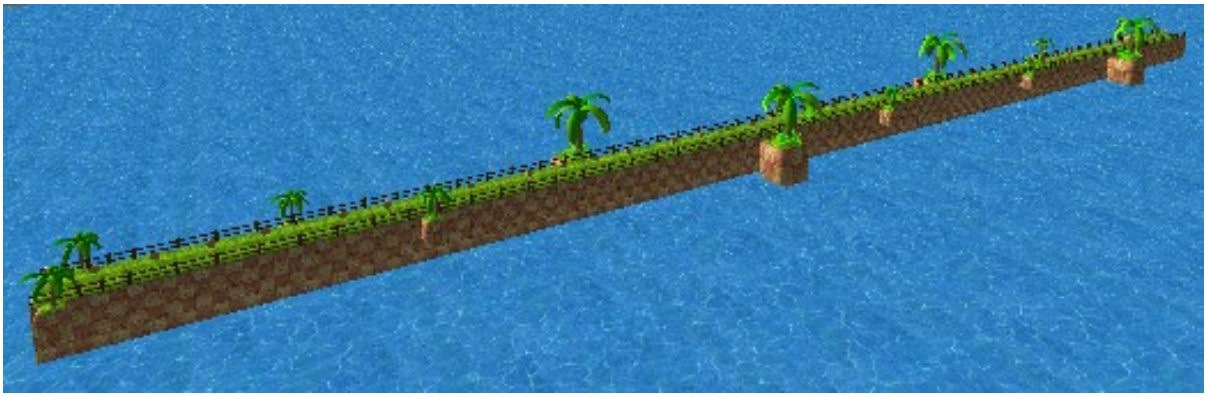

Figura 3.86 – Montagem completa da sequência de plataformas e do mar.

3.3.5.4.4. Marcação da distância percorrida e pontuação

Com uma proposta um pouco diferente dos demais jogos, o jogo "Corrida Infinita" traz um marcador de distância percorrida no jogo, podendo ser observado no canto superior esquerdo da Figura 3.70, com o marcador sendo do tipo "*Text*" acompanhado de uma imagem simbolizando um corredor, no seu lado esquerdo. O sistema de marcação obtém, a cada frame, o tempo desde o carregamento da cena atual, em segundos, e faz uma conversão, apenas visual, para metros passando a sensação ao jogador de imersão em uma corrida.

Este controle é feito utilizando o script "*Player.cs*", disponível no Apêndice S.

A pontuação é acrescida toda vez que o jogador coletar uma moeda através do avatar do jogo, assim quanto mais moedas o jogador coletar, maior será sua pontuação. Não há nenhum tipo de penalidade caso o usuário deixe de coletar uma das moedas fazendo com que o mesmo não se sinta pressionado a pegar todas as moedas, visando deixá-lo mais relaxado e evitando que ele faça movimentos bruscos.

Os dados salvos são adicionados às três últimas linhas do arquivo da Figura 3.16 quando o usuário clica no botão "salvar" do menu de pause do jogo.

3.4. Calibração

Nesta seção, será apresentado como é feita a calibração dos limites de movimentação dos jogos. Foi utilizada a cena "Calibração" para que o paciente realizasse os movimentos pedidos em frente à câmera. Estes valores são salvos em uma variável e utilizados nos jogos para a delimitação e ajuste dos limites de movimentação dos avatares de cada jogo.

3.4.1. Interface principal da cena "Calibração"

Nesta cena o paciente irá interagir com o output do corpo humano gerado pelo *OpenPose* na tela, onde o mesmo realiza todos os movimentos que o usuário faz em frente à câmera. A intensão da calibração é ter os limites de movimento do paciente para aplicar nas faixas de movimento utilizados pelos avatares nos jogos.

A Figura 3.87 mostra a tela padrão desta cena.

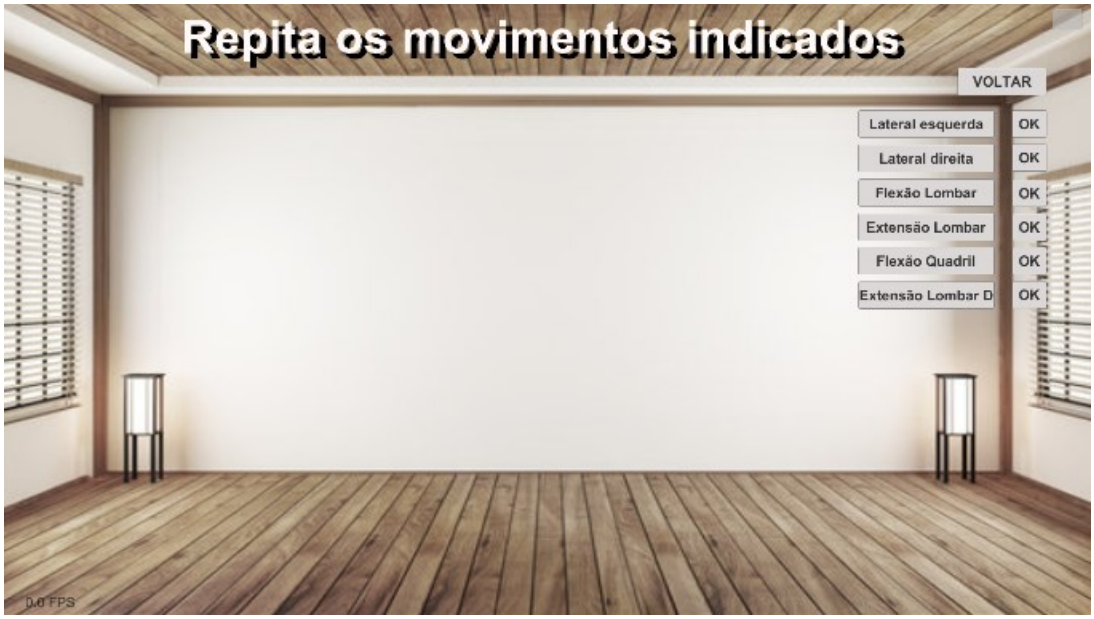

Figura 3.87 – Tela padrão da cena “Calibração”.

Pode ser observado na imagem que na parte direita estão contidos seis botões interativos com os nomes dos exercícios utilizados nos jogos e à frente de cada botão existe um outro pequeno botão escrito “OK”. Também nota-se através da Figura 3.85, uma frase na parte superior explicando o que o paciente deve fazer nesta cena.

Durante a calibração, o usuário terapeuta vai clicar em cada um dos seis botões com nomes de exercício, então será mostrado na tela, no canto inferior direito, um avatar de nome *Ybot*, que é um *Prefab* com a função de mostrar ao jogador qual é o movimento que ele deve realizar em frente a câmera. Cada botão representa um exercício diferente com seu *Ybot* mostrando como é aquele respectivo exercício.

A Figura 3.88 mostra o Prefab *Ybot*.

Figura 3.88– *Prefab Ybot*.

O modelo utilizado possui várias articulações e, devido a este fator, tornou mais entendível e melhor exemplificada a explicação de como devem ser realizados os movimentos de cada exercício. Foi feita uma animação com o *Ybot* para cada tipo de exercício.

Após a execução da animação do exercício escolhido pelo terapeuta o usuário repetirá os movimentos apresentados e ao atingir o limite de movimentação que seu corpo conseguir, para o referido exercício, o terapeuta ira clicar no botão "OK" do referido exercício. Ao efetuar essa ação o usuário terapeuta estará salvando as coordenadas cartesianas das articulações pré-definidas do paciente em uma variável, a qual será utilizada pelos *scripts* de cada avatar dos jogos, adaptando os limites de movimentação destes para a condição física de movimentação do paciente.

A Figura 3.89 mostra um exemplo de calibração para o exercício de flexão lateral esquerda com a pessoa se movimentando de acordo com o que é apresentado na animação do *Ybot* correspondente.

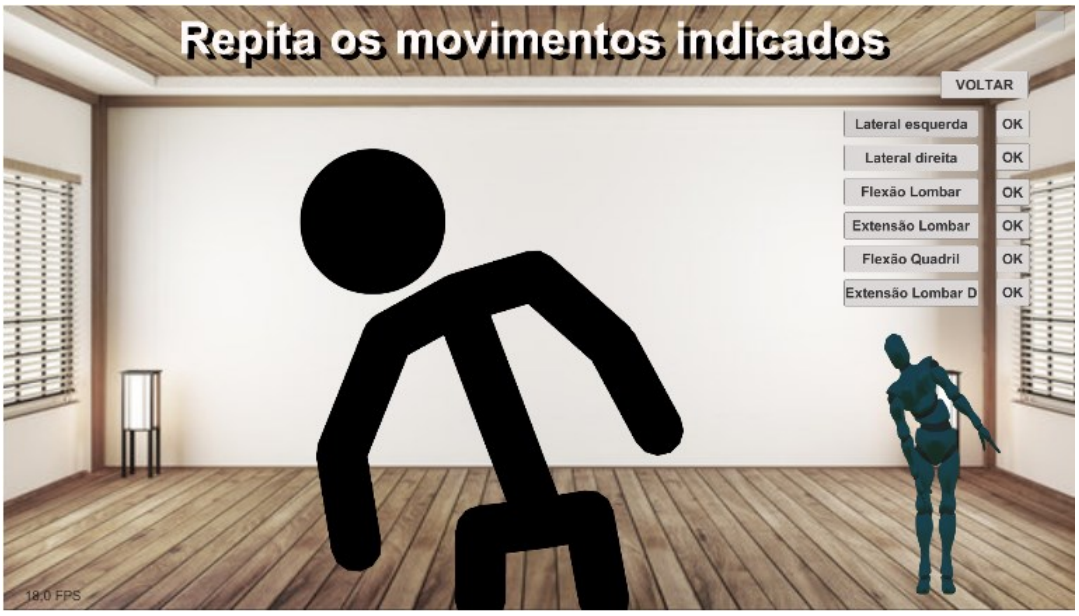

Figura 3.89 - Exemplo de calibração para o exercício de flexão lateral esquerda.

O *script* responsável por fazer o controle da calibração e salvar as coordenadas das articulações do paciente é o "botaocalib.cs", disponível no Apêndice V. O método utilizado por este *script* para fazer a comunicação dos valores salvos entre as cenas é utilizando o já citado *PlayerPrefs*.

Ao ser pressionado pelo usuário terapeuta, o botão "VOLTAR", no canto superior direito, retorna o mesmo à cena "Menu Principal".

As Figuras 3.90 e 3.91 mostram mais dois exemplos de calibração, são eles para o exercício de extensão lombar e para o de flexão lombar, respectivamente. Estas duas calibrações foram feitas com uma câmera auxiliar [câmera original de fábrica de um celular *Android*].

Figura 3.90 - Exemplo de calibração para o exercício extensão lombar (método McKenzie), posição deitado.

Figura 3.91 - Exemplo de calibração para o exercício flexão do quadril (método de Williams), posição deitado.

3.5. Métodos de controle

Aqui nesta seção, serão apresentados os métodos de controle utilizados nas principais etapas dos jogos. Serão mencionados os controles entre e dentro dos jogos, como os realizados entre cenas, controle da velocidade do avatar, alterações de cenário entre outros.

3.5.1. Mecanismo de controle dos menus

Nos menus, o principal mecanismo de controle é a existência de controle dos eventos entre as ações dos botões e preenchimento de caixas de texto. Sendo assim, a forma de controle destas ações será detalhada.

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

A Figura 3.92 mostra esta parte do componente em questão.

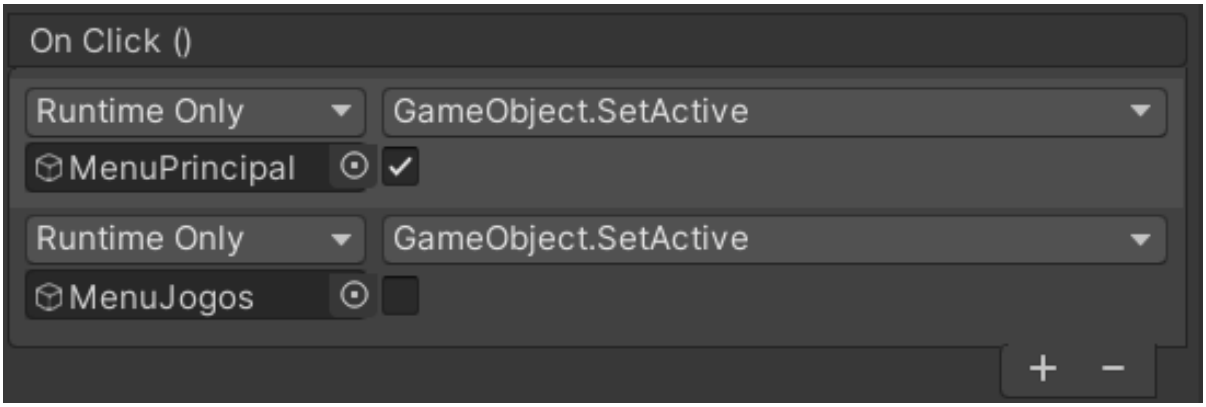

Figura 3.92 – Configuração "*On Click ()*" dentro do componente "*Button (Script)*".

Com a possibilidade de se utilizar este recurso de realizar uma ação ao clique do botão, desenvolveu-se scripts de alternância entre os jogos, as cenas, telas, de saída dos jogos e até mesmo para salvar as informações dos pacientes e dos jogos.

O exemplo dado acima é do botão voltar existente na cena "MenuJogos" e a ação descrita como *"GameObject.SetActive"* ativa a cena "MenuPrincipal" e desativa a cena "MenuJogos" através do clique no botão. Esta sequência de eventos representa a hierarquia das ações de ativar e desativar as duas cenas em questão.

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 3.93 mostra a janela para organização das cenas na construção do jogo.

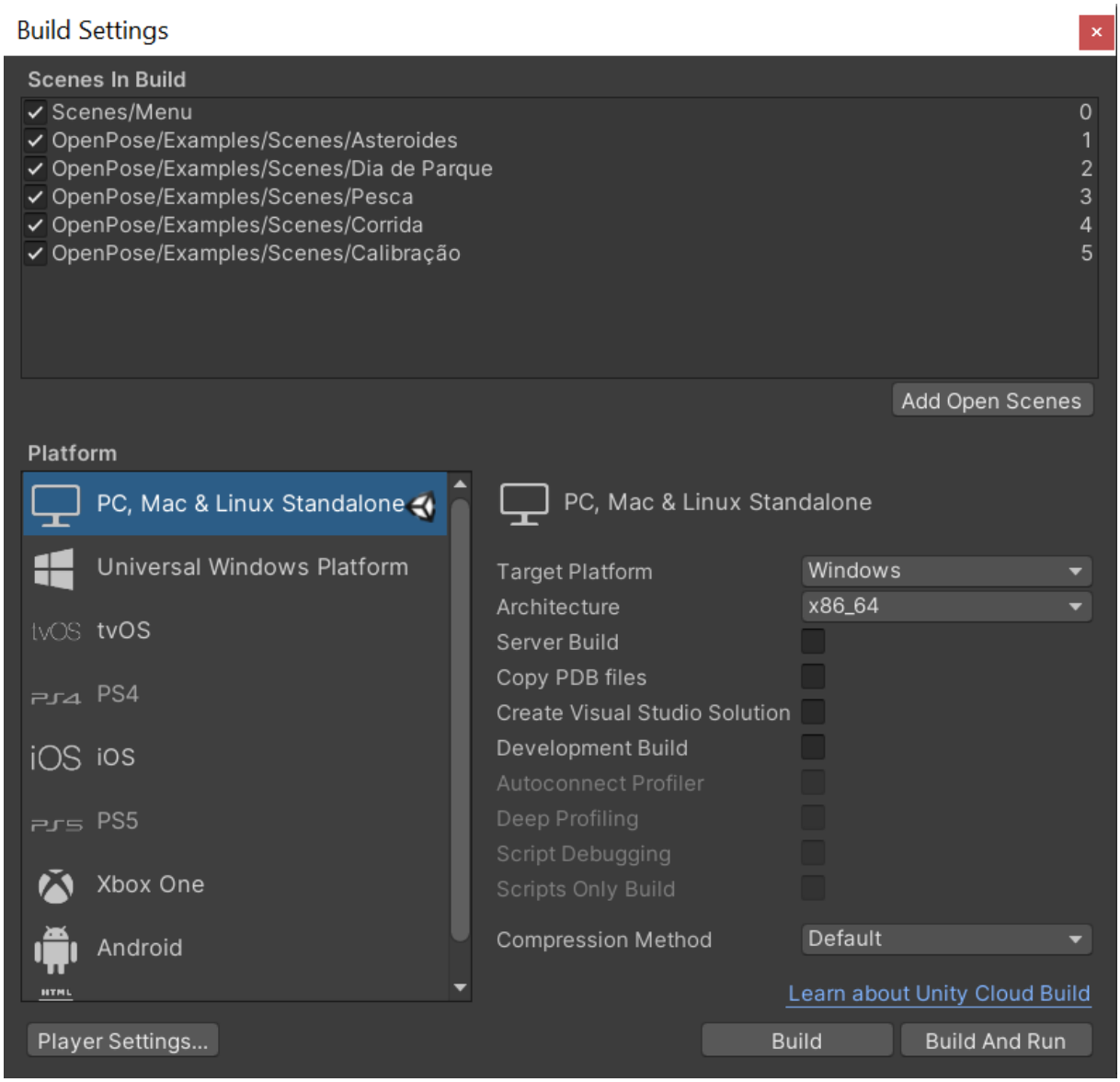

Figura 3.93 – Janela "*Build Settings*", aqui são colocadas as cenas na ordem de construção do jogo.

Não é necessário um *script* específico para se executar a ação responsável por alternar telas em uma mesma cena. Toas as alternâncias entre cenas foram feitas através das ações pré-estabelecidas do organizador de eventos *On Click ()*, sem a necessidade de um *script* específico para estas ações.

Por fim, a ação de se sair dos jogos com o clique no botão "*QuitButton*" da cena "*PauseMenu*" é gerada a partir do script "*PauseMenu.cs*", no Apêndice W, responsável por

executar a ação da aplicação. Isto é possível pois é acessado o método “SairMenu ()”, cujo qual finaliza a seção.

3.5.2. Controle dos avatares

O controle dos avatares dos jogos já foi abordado na seção 3.33, aqui será detalhado como é feita a transcrição dos movimentos, realizados pelo jogador, para os jogos através do *OpenPose*.

O *OpenPose* utiliza diversos pontos de análise que representam articulações do corpo humano. Nos jogos foram utilizadas duas articulações específicas para o controle de movimentos dos avatares, são elas a articulação do pescoço (ponto 1) e a articulação central do quadril (ponto 8). A Figura 3.94 mostra os pontos utilizadas pelo *OpenPose*.

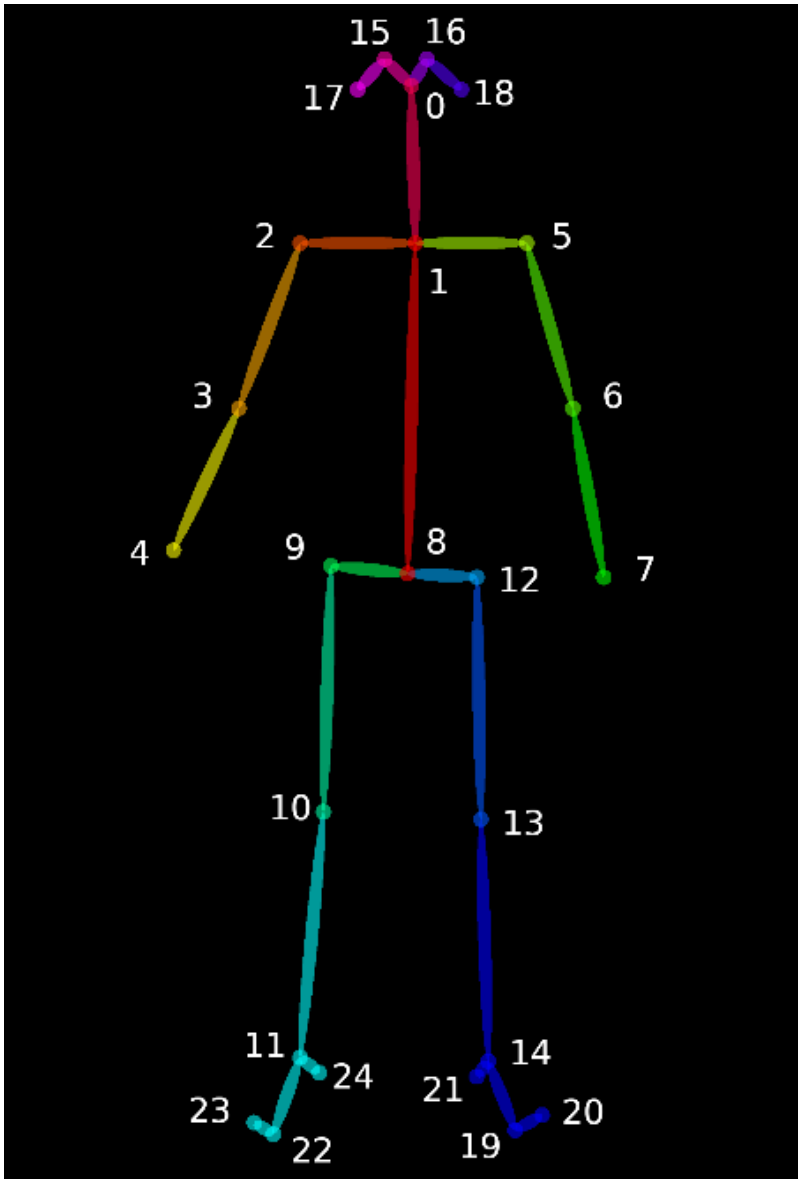

Figura 3.94 - Pontos utilizados pelo OpenPose para detecção das articulações do corpo humano.

A Figura 3.95 mostra um exemplo de como é o *output* do corpo humano que o *OpenPose* apresenta na tela, para uma pessoa fazendo movimentos em frente a câmera. Esta imagem foi gerada na cena de "Calibração".

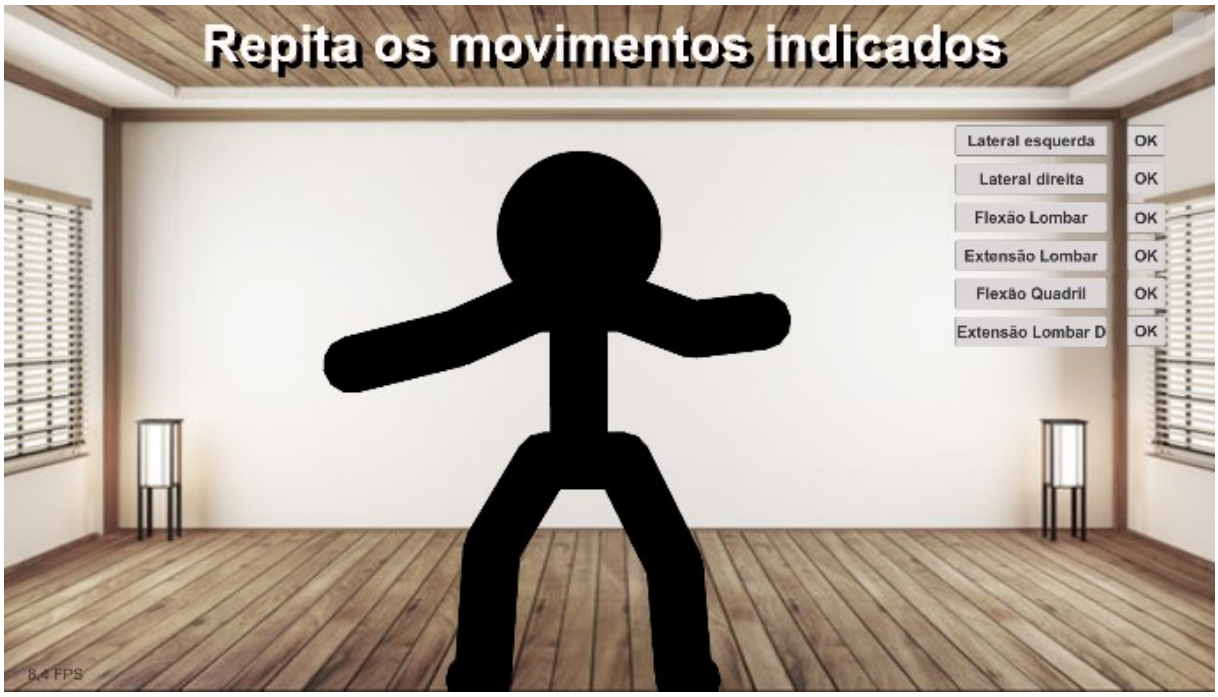

Figura 3.95 – *Output* do corpo humano feito pelo *OpenPose* na cena "Calibração".

Depois de identificadas as juntas, tem-se as informações das posições (x,y,z) das mesmas dentro dos jogos a cada frame. Assim, estes valores podem ser utilizados para se movimentar o avatar de acordo com os movimentos do usuário, com a mínima perda possível. Para os jogos em questão o atraso (*delay*) apresentado inicialmente foi perceptível e então foram feitos ajustes nos jogos e nas configurações.

Mudou-se o método de *spawn* das animações para o que está contido nos *scripts* encontrados nos Apêndices I, J, K, P e T. Também melhorou-se os códigos que fazem tal controle para evitar repetições de comandos em ações parecidas.

Outra alteração feita foi a configuração para o valor "80" em "*Net Resolution*" na janela "*Inspector*" do objeto "*OpenPose"* encontrado na janela "*Hierarchy*".

Após realizados os ajustes o atraso nos jogos passou a ser imperceptível tornando possível a utilização destes de maneira muito satisfatória.

Os scripts utilizados para os controles dos avatares e demais componentes dos jogos estão citados nos itens 3.3.5.1., 3.3.5.2., 3.3.5.3. e 3.3.5.4. cada um em seu respectivo jogo.

O OpenPose utiliza o script *"OpenPoseUserScript.cs"* para tratar as imagens recebidas pela câmera do computador e localizar os pontos de articulações do corpo humano, gerando assim o esqueleto mostrado na Figura 3.81. O *Prefab* do corpo humano com todas as articulações e pontos que o *OpenPose* utiliza é o *"Human2D_Body25"*, o qual teve o nome alterado para cada jogo e para a cena de "Calibração", partindo do *"Human2D_Body25" até o "Human2D_Body29"*.

CAPÍTULO IV

RESULTADOS E DISCUSSÕES

Os resultados obtidos com os jogos, e os menus desenvolvidos, se mostraram satisfatórios, uma vez que atenderam todos os requisitos inicialmente levantados para o projeto. Através da utilização do motor de jogos Unity 3D foi possível um melhor e mais facilitado desenvolvimento de jogos esteticamente agradáveis, suaves e atrativos, visando sempre a manutenção da essência da seriedade que é uma reabilitação.

O Apêndice A mostra o passo a passo completo da criação de um dos jogos. Através deste Apêndice é apresentada a utilização das ferramentas de auxílio dentro do motor de jogos dispensando, quando possível, a programação pura para controle dos objetos. Durante o desenvolvimento deste jogo em questão, assim como nos demais, procurou-se criar scripts simples e intuitivos sempre que possível, uma vez que os objetos são de fácil referência dentro do jogo.

Porém, essa facilidade certas vezes torna-se um limitador para certas ações e até torna elas difíceis de serem implementadas quando o motor de jogos Unity 3D insiste em auxiliar procedimentos em algumas situações complexas, como por exemplo na utilização dos gatilhos juntamente com 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. Durante o desenvolvimento do jogo teste de nome “Estrelas”, isto fez com que o objeto de jogo “Avatar”, representado por uma plataforma girante que coletava estrelas cadentes, tivesse que ser modificado do padrão, perdendo seu contato único de colisão ao longo de sua superfície e transformando-o em várias colisões.

Isto fazia com que a colisão acontecesse mais de uma vez e levasse a estrela até o local de coleta, resolvendo o problema que acontecia com apenas uma colisão, que era o fato de a estrela “atravessar” a plataforma após a colisão acontecer.

A partir do jogo teste foram modificados os avatares utilizados nos quatro jogos descritos neste trabalho, tornando o funcionamento dos mesmos satisfatório e de acordo com o planejado inicialmente.

Também durante o desenvolvimento do jogo teste descobriu-se que alguns exercícios típicos de reabilitação da coluna não poderiam ser abordados no presente trabalho devido às limitações que *OpenPose* tem, como a existência de apenas dois pontos de articulação na coluna, e à barreira tecnológica que a inclusão destes exercícios impunha sob o projeto, como por exemplo a utilização de 2 ou mais câmeras auxiliares por exercício e mesmo ao posicionamento destas câmeras que o exercício exigia, tornando inviável sua aplicação.

Alguns exemplos de exercícios que não puderam ser abordados neste trabalho são os exercícios que exigiam a torção da coluna em torno de seu próprio eixo e o exercício popularmente chamado de “gato arrepiado”, onde o paciente mobiliza toda a coluna para cima lentamente e depois para baixo com as mãos e joelhos apoiados no chão.

A partir dos dados coletados do jogo teste, fez-se um estudo, com o apoio de um fisioterapeuta, de quais exercícios seriam escolhidos, chegando-se aos 6 escolhidos e já abordados nesta monografia. A divisão dos exercícios, e dos seus tempos implementados nos jogos, foi feita de maneira a facilitar a prática dos mesmos pelo paciente e incentiva-lo à completar os exercícios de maneira correta.

Além da impossibilidade de se utilizar alguns exercícios fisioterápicos nos jogos, também foram encontradas algumas outras dificuldades durante os seus desenvolvimentos.

Ao se trabalhar com as animações, encontradas nos jogos, notou-se que o método de spawn inicialmente implementado era inviável, pois seu custo computacional era altíssimo, chegando a fazer o travamento do computador utilizado e finalizando o software por falta de resposta. Este problema foi resolvido com otimização do *script* e nos casos menos exigentes em que uma grande otimização não era necessária, o *spawn* da animação era realizado e sua destruição era feita ao ser finalizada. Um exemplo desta otimização foi a já citada “reciclagem” das plataformas no jogo “Corrida Infinita”.

Outro fator a ser comentado é o método de pontuação inicialmente implementado. Em alguns casos ele poderia passar a impressão ao jogador de que o mesmo estava tendo algum tipo de penalidade caso não conseguisse completar o necessário para fazer algum dos pontos.

Este problema foi completamente resolvido e tudo que pudesse passar esta impressão ao jogador foi retirado dos jogos, como por exemplo os asteroides que chegavam a colidir com a nave que o jogador controlava caso não fossem destruídos a tempo, o que poderia gerar um movimento brusco, por parte do paciente, para desviar a nave. Resolveu-se este problema fazendo com que o asteroide surgisse lentamente na tela e mesmo que não fosse destruído, o mesmo para em uma posição longe da nave e fica à espera de um novo posicionamento da nave para sua destruição.

Outro exemplo foi o fato de inicialmente existirem obstáculos na ponte do jogo “Corrida Infinita”, mesmo que o paciente não perdesse pontos ao colidir com tais obstáculos, o mesmo poderia realizar algum movimento brusco na tentativa de desviar dos mesmos. Resolveu-se este problema retirando os obstáculos e ao invés de o usuário receber pontos por desviar dos já retirados obstáculos, ele passou a receber pontos por coletar moedas que vão surgindo na tela à medida que o avatar “robô”, controlado pelo usuário percorre a ponte.

Para a captura das imagens em tempo real foi utilizada a câmera *webcam* embutida originalmente no notebook e também, como câmera auxiliar necessária em alguns exercícios fisioterápicos, a câmera de um celular com sistema operacional Android e um software de computador para fazer a comunicação entre o celular e o notebook. O software utilizado para tal comunicação é o *DroidCamApp*, disponível gratuitamente para download na internet.

Não foi possível se utilizar um celular com sistema operacional IOS pois a versão gratuita do software utilizado não possui está opção de comunicação.

O computador utilizado para o desenvolvimento dos jogos é um Notebook Samsung com as seguintes configurações básicas:

* Processador Intel core i7 sétima geração;
* Placa de vídeo NVIDIA GeForce 940MX de 2 GB;
* Memória RAM de 8 GB;
* HD de 1 TB;
* Webcam da marca Microsoft (original de fábrica do Notebook).

Durante a realização deste trabalho foram realizados testes de jogabilidade em outros três computadores, dois com melhores configurações, outro com configurações parecidas às do Notebook Samsung e um último com configurações mais básicas.

O computador mais potente utilizado tem configurações:

* Processador Intel core i7 sétima geração;
* Placa de vídeo GTX1080 Zotac de 8GB;
* Memória RAM de 16 GB;
* HD de 1 TB;

Outro computador utilizado tem configurações:

* Processador Intel core i3 nona geração;
* Placa de vídeo NVIDIA GTX1650TI de 4 GB;
* Memória RAM de 8 GB;
* HD de 1 TB;

O último computador utilizado foi o notebook Acer Aspire 3 de configurações:

* Processador Intel core i3 quinta geração;
* Placa de vídeo integrada Intel Graphics 620;
* Memória RAM de 8 GB;
* HD de 1 TB;

Os resultados obtidos foram que, no computador de menor configuração os jogos com maior custo computacional apresentaram um certo *delay* durante suas execuções, em momentos de pico de uso do computador, porém não foram grandes o suficiente para atrapalhar a seção de jogos e a experiência do usuário. Já nos demais computadores, não notou-se nenhum *delay* e os jogos rodaram de maneira suave e contínua sem nenhum tipo de travamento, observou-se no computador de melhores configurações uma ligeira melhora em relação ao Notebook utilizado para o desenvolvimento.

A Figura 4.1 mostra a tela de funcionamento do jogo “Asteroides” e ao lado uma foto do jogador realizando o movimento de extensão lombar na posição deitado. Observa-se que o avatar do jogo acompanha o movimento realizado pelo jogador.

Figura 4.1 - Exemplo do jogador controlando o avatar do jogo “Asteroides” através da coluna.

A Figura 4.2 mostra a tela de funcionamento do jogo “Dia de Parque” e ao lado uma foto do jogador realizando o movimento de flexão lombar. Observa-se que o avatar do jogo acompanha o movimento realizado pelo jogador.

Figura 4.2 – Exemplo do jogador controlando o avatar do jogo “Dia de Parque” através da coluna.

A Figura 4.3 mostra a tela de funcionamento do jogo “Pesca” e ao lado uma foto do jogador realizando o movimento de flexão do quadril na posição deitado. Observa-se que o avatar do jogo acompanha o movimento realizado pelo jogador.

Figura 4.3 - Exemplo do jogador controlando o avatar do jogo “Pesca” através da coluna.

A Figura 4.4 mostra a tela de funcionamento do jogo “Corrida Infinita” e ao lado uma foto do jogador realizando o movimento de flexão lateral esquerda. Observa-se que o avatar do jogo acompanha o movimento realizado pelo jogador.

Figura 4.4 - Exemplo do jogador controlando o avatar do jogo “Corrida Infinita” através da coluna.

Abaixo seguem os *links* dos vídeos explicativos respectivos ao funcionamento de cada jogo.

- Asteroides:
  https://youtu.be/xxY53ApRHPQ
- Dia de Parque:
  https://youtu.be/nkJ11Lh6YPU
- Pesca:
  https://youtu.be/UR-hfVx1atA
- Corrida Infinita:
  https://youtu.be/iT0eamlOe8s

CAPÍTULO V

CONCLUSÃO

Com a realização deste projeto foi possível desenvolver jogos sérios com o intuito de auxiliarem na reabilitação da coluna humana e, desta forma, espera-se a melhora dos processos de reabilitação para estes usuários principalmente no que diz respeito ao engajamento e continuação dos tratamentos.

O planejamento e escolha dos exercícios abordados foi uma das principais atividades, uma vez que, apesar de ser um instrumento de criação lúdica, o mesmo tem como objetivo final auxiliar no tratamento de reabilitação dos pacientes, cada detalhe na elaboração do planejamento foi crucial para que seu objetivo seja alcançado na prática.

Para se alcançar o produto final deste projeto foi importante a utilização de uma versão gratuita do motor de jogos Unity 3D juntamente com o *OpenPose*, tanto na criação e desenvolvimento dos jogos, quanto na experiência e resultados obtidos, cujos quais podem ser utilizados para o aprimoramento de técnicas de auxílio em tratamentos de saúde, como foi para a reabilitação da coluna humana, e futuramente para outros tipos de reabilitação. A facilidade de desenvolvimento e integração entra as duas plataformas utilizadas torna possível o desenvolvimento de jogos sérios com possibilidade de serem mais elaborados e com gráficos ainda melhores em projetos futuros.

Os cenários foram desenvolvidos para passar ao usuário a sensação de imersão dentro dos jogos, no intuito de aumentar o interesse do paciente em dar continuidade ao tratamento e fazer os exercícios corretamente. Tudo isto foi feito utilizando características visuais e sonoras que passassem uma sensação de relaxamento e segurança para os usuários, uma vez que tratamentos de reabilitação podem envolver questões sensíveis e, muitas vezes, dolorosas para os pacientes.

Alguns pontos dentro dos jogos não puderam ser melhor desenvolvidos devido à questões de limitação de captação das imagens para a análise do OpenPose, uma vez que só conseguiu-se utilizar uma câmera por vez durante a execução dos jogos e o posicionamento do usuário/paciente em relação à câmera [webcam ou auxiliar] encobre algumas articulações, não permitindo que estas pudessem ser utilizados na codificação de movimentação dos avatares.

Este fato fez com que alguns movimentos planejados para os avatares não pudessem ser realizados de maneira eficaz, movimentos estes que envolviam o giro da coluna e sobreposição de pontos chave, como o alinhamento da coluna perpendicular ao plano da câmera. Então optou-se por diminuir o número de articulações que seriam utilizadas, chegando às duas citadas no trabalho, que são o pescoço e a parte central do quadril.

O fato do OpenPose não possui um ponto de articulação entre o pescoço e o quadril também fez com que o leque de opções de exercícios diminuísse durante o planejamento do projeto.

Porém estes fatores não acarretaram em problemas nem comprometeram o desenvolvimento e execução dos jogos.

A boa arquitetura do motor de jogos Unity 3D através da utilização dos mecanismos de criação dos painéis de interface do usuário (UI), trouxe grande facilidade na criação de menus e também da tela de inserção de informações do usuário, além da facilidade de se colocar informações de desempenho durante a execução dos jogos.

O fato de não existirem condições de parada nem métodos de punição em casos de não coleta de pontos, permite com que os profissionais da área de reabilitação possam acompanhar a melhora quantificada no desenvolvimento dos seus pacientes enquanto os mesmos não se sentem frustrados com o jogo, caso não consigam executar determinadas tarefas, o que espera aumentar o engajamento e interesse destes pacientes e torna mais fácil o acompanhamento dos resultados por parte dos profissionais responsáveis.

Com a realização deste projeto foi possível se observar os mecanismos de desenvolvimento de um jogo e como isto pode ser implementado em várias outras áreas de atuação e, não somente, para o entretenimento. Também observou-se as diversas maneiras e locais de pesquisa que podem ser utilizados para a resolução dos problemas e dificuldades que possam vir a surgir em projetos similares.

REFERÊNCIAS BIBLIOGRÁFICAS

INSTITUTO BRASILEIRO DE GEOGRAFIA E ESTATÍSTICA. Pesquisa Nacional de Saúde. Percepção do estado de saúde, estilo de vida, doenças crônicas e saúde bucal, 2019. Rio de Janeiro: Instituto Brasileiro de Geografia e Estatística; 2020.

SILVEIRA, R., CERDEIRA, D. Algias na coluna vertebra: A abordagem fisioterapêutica de discentes de uma instituição de ensino superior em Sobral – Ceará. Sobral, 2014.

OLIVEIRA, C., SOUZA, D., MAGALHÃES, A., SILVA, J., CORREIA, G. Prevalência e fatores associados a problema crônico de coluna em mulheres em idade fértil. Trairi, 2015.

GARCIA, A., GONDO, F., COSTA, R., CYRILLO, F., COSTA, L. Efeitos de duas intervenções fisioterapêuticas em pacientes com dor lombar crônica não-específica: viabilidade de um estudo controlado aleatorizado. São Carlos, 2011.

HAGEN KB, TAMBS K, BJERKEDAL T. A prospective cohort study of risk factors for disability retirement because of back pain in the general working population. Spine. 2002 Aug; 27(16):1790-6.

SIQUEIRA FV, FACCHINI LA, HALLAL PC. Epidemiology of physiotherapy utilization among adults and elderly. Rev Saúde Pública. 2005; 39(4):663-8.

KISNER C, COLBY LA. Exercícios terapêuticos: fundamentos e técnicas. 3ª ed. São Paulo: Manole; 2005.

DIAS, R., SAMPAIO, I., TADDEO, L. Fisioterapia x Wii: A Introdução do Lúdico no Processo de Reabilitação de Pacientes em Tratamento Fisioterápico. In: Simpósio Brasileiro de Jogos e Entretenimento Digital, 8., 2009, Rio de Janeiro. Pôsteres da trilha de cultura. Rio de Janeiro: [s.n.], 2009.

HIDALGO, G., CAO, Z., SIMON, T., WEI, S.-E., JOO, H., SHEIKH, Y. “OpenPose library”, 2019. Disponível em <https://github.com/CMU-Perceptual-Computing-Lab/openpose>. Acesso: 12 set. 2021.

FANG, H.-S., XIE, S., TAI, Y., LU, C. “AlphaPose library”, 2017. Disponível em: <https://github.com/MVIG-SJTU/AlphaPose >. Acesso em: 12 set. 2021.

CLAUSS, C., ABDULLA, W. “Mask_RCNN library”, 2017. Disponível em: <https://github.com/matterport/Mask_RCNN>. Acesso em 12 set. 2021.

MACEDO, V., SANTOS, J. “*Análise Cinemática Automática usando OpenPose e Dynamic Time Warping com Aplicações no Remo*”, 2019. Disponível em: <https://bdm.unb.br/bitstream/10483/24937/1/2019_VictorOliveiraMacedo_JoyceDaCostaSantos_tcc.pdf > Acesso em: 12 set. 2021.

SILVA, M. *Human Action Recognition Based On Spatiotemporal Features From Videos*. São Carlos. 2020. 88p. Dissertação (Pós-Graduação em ciência da computação) – Universidade Federal de São Carlos. São Carlos, 2020. [Orientador: Prof. Dr. Aparecido Nilceu Marana].

NEWZOO. The Brazilian Gamer 2017. Disponível em <https://newzoo.com/insights/infographics/the-brazilian-gamer-2017/>. Acesso: 12 set. 2021.

BARNES T., MIGUEL ENCARNAÇÃO L., SHAW D. (2009) “Serious Games”, IEEE Computer Graphics and Applications 29(2), p. 18-19.

MACHADO, L.S.; MORAES, R.M.; NUNES, F. (2009) “Serious Games para Saúde e Treinamento Imersivo”. Book Chapter. In: Fátima L. S. Nunes; Liliane S. Machado; Márcio S. Pinho; Cláudio Kirner. (Org.). Abordagens Práticas de Realidade Virtual e Aumentada. Porto Alegre: SBC, p. 31-60.

NETTER, Frank H.. Atlas de Anatomia Humana. 2ed. Porto Alegre: Artmed, 2000.

Natour, Jamil. “Coluna vertebral”. 2º ed. São Paulo, 2004.

Magalhães, Lana. Coluna Vertebral. Disponível em: https://www.todamateria.com.br/coluna-vertebral. Acesso em 10 de set. de 2021.

Antunes M, Favoreto A. Análise comparativa dos efeitos da massoterapia e pompage cervical na dor e qualidade de vida em mulheres. ConScientiae Saúde. 2017 Mar; 16(1): 109-115.

Bring G, Bring J: Neck pain in general population. Spine 20:624-7, 1995.

ANTONIO, SILVIO F. “Diagnóstico Diferencial das Cervicalgias”. Coluna vertebral. 2º ed., p. 47 – 48, São Paulo, 2004.

FREIRE, MARLENE. “Diagnóstico Diferencial das Cervicalgias”. Coluna vertebral. 2º ed., p. 77 - 92, São Paulo, 2004.

Deyo RA, Phillips WR. Low back pain. A primary care challenge. Spine 21: 2826-32, 1996.

Cecin HA. Proposição de uma reserva anatomofuncional, no canal raquidiano, como fator interference na fisiopatologia das lombalgias e lombociatalgias mecânico-degenerativas. Rev Assoc Med Bras 43:295-310. 1997.

Nachemson AL. Newest knowledge of low back pain. A critical look. Clin Orthop 279:8-20, 1992.

Shcon RP et al. Soft tissue rheumatic pain. 3rd ed; 1996. p. 391.

Bigos SJ, Battie MC, Spengler DM, et al. A prospective study of work perceptions and psychosocial factors affecting the report of back injury. Spine 16:1-6, 1991.

Leboeuf-Y de C, Kyvik KO, Bruun NH. Low back pain and lifestyle. Part II. Obesity. Information from a population-based sample of 29, 424 twin subjects. Spine 24:779-83, 1999.

Deyo RA, BassJE. Lifestyle and low-back pain. The influence of smoking and obesity. Spine 14:501-6. 1989.

Cecin HA. et al. Dor lombar e trabalho pesado: aspectos epide-miológicos. Rev Bras Reumatol 32:157-62, 1992.

Leino P,Magni G. Depressive and distress symptoms as predictors of low backpain, neck-shoulder pain, and other musculoskeletal morbidity: a 10 years follow-up of metal industry employees. Pain 53:89-94, 1993.

Heliovaara M, Makela M, Knekt P, et al. Determinants of sciatica and low-back pain. Spine 16:608-14, 1991.

McGorry RW, Hsiang SM, Snook SH, et al. Meteorological conditions and self-report of low back pain. Spine 23:2096-102, 1998.

Brazil, AV et al. Diagnóstico e tratamento das lombalgias e lombociatalgias. Ver. Brass. Reumatol., 2004.

Vialle LR, Vialle EM, Henao JE, Giraldo G. Hérnia discal lombar. Rev Bras Ortop. 2010;45(1):17-22.

Barros Filho TE, Basile Júnior R, Cristante AF, Araújo MP. Coluna toracolombar: síndromes dolorosas. In: Hebert S, Barros Filho TE, Xavier R, editors. Pardini Júnior AG. Ortopedia e traumatologia: princípios e prática. 4a ed Porto Alegre: Artmed; 2009. p. 122-35.

Deyo RA, Loeser JD, Bigos SJ. Herniated lumbar intervertebral disk. Ann Intern Med. 1990;112(8):598-603.

Stienen MN, Cadosch D, Hildebrandt G, Gautschi OP. The lumbar disc herniation - Management, clinical aspects, and current recommendations. Praxis (Bern 1994). 2011;100(24):1475-85.

Mixter WJ, Barr JS. Rupture of intervertebral disc with involvement of the spinal canal. N Engl J Med. 1934;211:210-4.

Revel M, Amor B: Les sciatiques en dehors de la hernie discale.. Rev Prat 42:549-53, 1992.

RADU, ARI S. “Hérnia Discal”. Coluna vertebral. 2º ed., p. 155 – 162, São Paulo, 2004.

CAILLIET, R. Escoliose diagnóstico e tratamento. São Paulo: Manole, 1979.

LAPIERRE, A. La reeducacion física. Barcelona: Ed. Científico-Médica,1977.

FERREIRA, WANDA .H. R. “Escolioses e Alterações Posturais”. Coluna vertebral. 2º ed., p. 165 – 187, São Paulo, 2004.

Moe, Winter, Bradford, Lonstein. Scoliosis and other spinal Deformities. Philadelfia: WB Saunders, 1978.

Winter R: Adolescent idiopathic scoliosis. N Engl J Med 314:1379, 1986.

Bradford D: Neurological complications in Scheuermann‘s disease. J Bone Join Surg 51-A:657, 1969.

Kendall, McCreary: Muscles: Testing and Function. 3 ed. Baltimore: Williams & Wilkins, 1983.

Walker, Rothstein: Relationships between lumbar lordosis, pelvic tilt and abdominal performance. Physical Therapy 67:512, 1987.

Aebi M. The adult scoliosis. Eur Spine J 2005;14(10):925–948

Youssef JA, Orndorff DO, Patty CA, et al. Current status of adult spinal deformity. Global Spine J 2013;3(01):51–62

Vernon-Roberts B, Moore RJ, Fraser RD. The natural history of age related disc degeneration: the influence of age and pathology on cell populations in the L4-L5 disc. Spine (Phila Pa 1976) 2008;33 (25):2767–2773.

Silva FE, Lenke LG. Adult degenerative scoliosis: evaluation and management. Neurosurg Focus 2010;28(03):E1.

Marty-Poumarat C, Scattin L, Marpeau M, Garreau de Loubresse C, Aegerter P. Natural history of progressive adult scoliosis. Spine (Phila Pa 1976) 2007;32(11):1227–1234.

Natour, Jamil. “Reabilitação e Coluna Vertebral”. Coluna vertebral. 2º ed., pq. 207 – 212, São Paulo, 2004.

DePalma MJ, Slipman CW. Evidence-informed management of chronic low back pain with epidural steroid injections. Spine J 2008;8(01):45–55.

Haldeman S, Nordin M, Chou R, et al. The Global Spine Care Initiative: World Spine Care executive summary on reducing spine-related disability in low- and middle-income communities. Eur Spine J 2018;27(Suppl 6):776–785.

Passias PG, Jalai CM, Line BG, et al. International Spine Study Group. Patient profiling can identify patients with adult spinal deformity (ASD) at risk for conversion from nonoperative to surgical treatment: initial steps to reduce ineffective ASD management. Spine J 2018;18(02):234–244.

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.

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.

Bezerra, Anne de Souza. Uma revisão acerca dos métodos: Mackenzie vs.Williams. 2016, p. 6 – 15.

APÊNDICES

A – TUTORIAL PARA DESENVOLVIMENTO DO JOGO “ASTEROIDES”

Neste apêndice, será apresentado um tutorial para desenvolvimento do jogo “Asteroides”. Serão apresentadas as informações em textos e imagens do próprio software, com uma distribuição feita em etapas, desde a obtenção do arquivo OpenPose até a finalização do jogo.

A versão utilizada do Unity 3D é a 2019.4.24f1.

1. Acesse o endereço abaixo e baixe o arquivo contendo os componentes do *OpenPose*:

   <https://github.com/CMU-Perceptual-Computing-Lab/openpose_unity_plugin>

   Após baixado o arquivo, instale-o conforme as instruções contidas em:

   <https://github.com/CMU-Perceptual-Computing-Lab/openpose_unity_plugin/blob/master/doc/installation.md>

2. Por uma questão de simplificação o jogo foi desenvolvido a partir do próprio arquivo de demonstração que é baixado juntamente ao OpenPose no passo 1. Apaga-se o conteúdo do arquivo deixando somente os componentes do OpenPose e então inicia-se o desenvolvimento do jogo renomeando-se a cena principal para o nome escolhido. A Figura A.1 mostra como é a aparência do programa ao se abrir o arquivo “*demo*” do *OpenPosePlugin*.

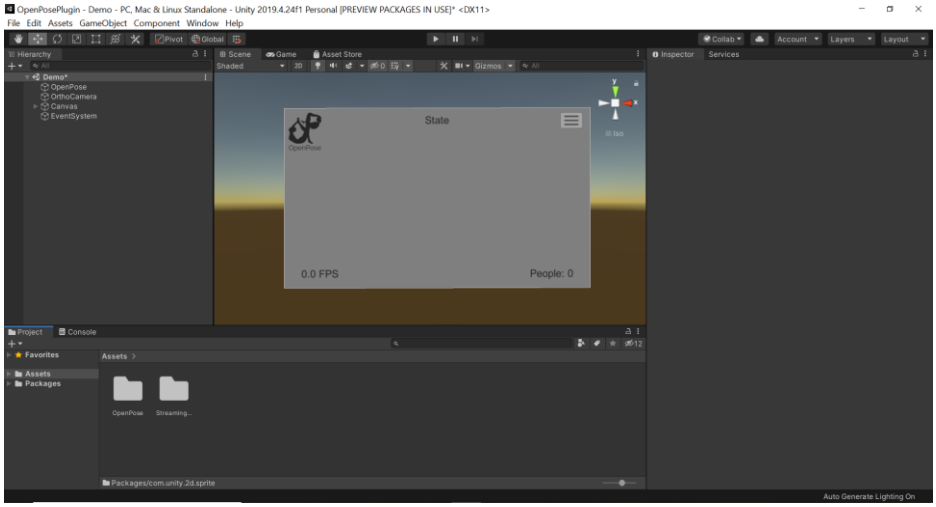

Figura A.1 – Tela inicial do Unity ao se abrir o arquivo de demonstração do *OpenPosePlugin*.

3. No canto superior direito, clique em “Layout” e mude o *layout* para “2 by 3”, conforme mostra a Figura A.2;

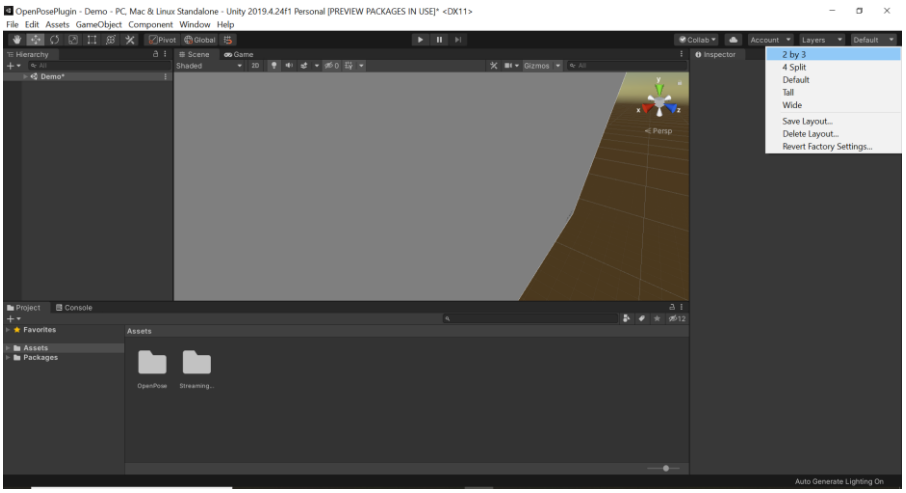

Figura A.2 – Interface principal do Unity 3D.

4. Reorganize o novo layout arrastando as janelas conforme mostra a Figura A.3;

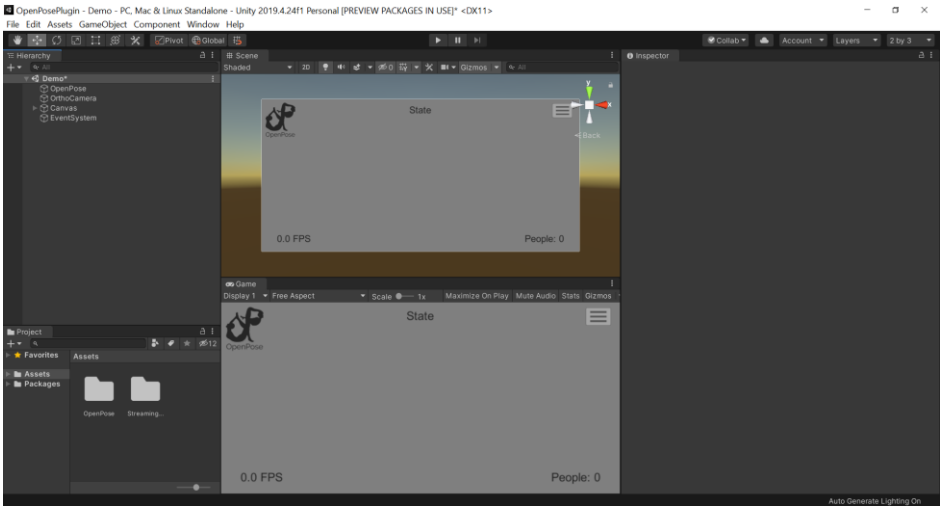

Figura A.3 – Interface modificada do Unity 3D.

5. Caso deseje salvar o novo layout, clique em “Save Layout...” na mesma caixa de controle do passo 3;
6. Desative os componentes pertencentes a Interface do Usuário(UI), na janela “Hierarchy”, conforme indicado na Figura A.4. Para isto clique na caixa de seleção correspondente de cada elemento na janela “Inspector”, conforme a Figura A.5.

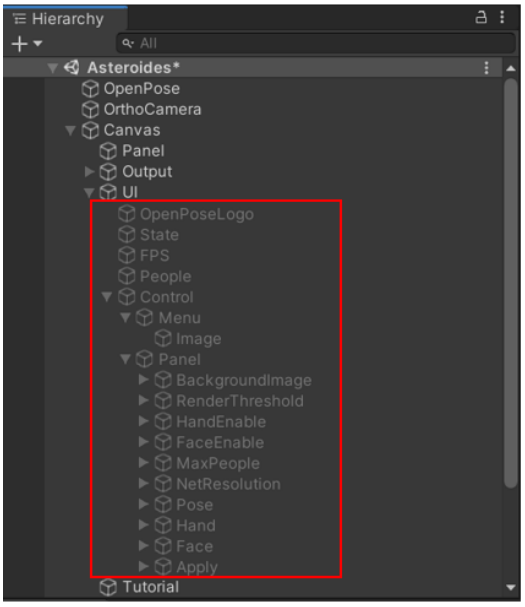

Figura A.4 – Componentes da UI a serem desativados.

Figura A.5 – Desativando o elemento na janela “Inspector”.

7. O painel na janela “Scene” deverá ficar semelhante ao apresentado na Figura A.6.

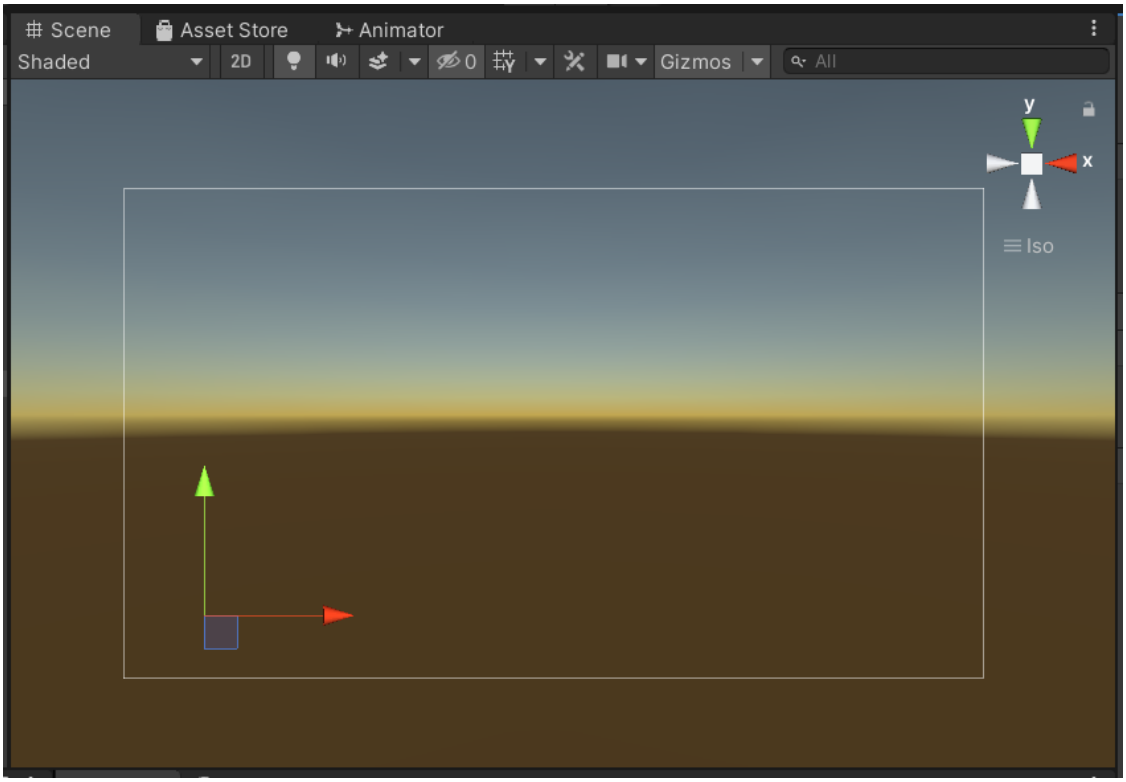

Figura A.6 – Painel após as modificações.

8. Crie uma pasta dentro da pasta raiz que contém o arquivo que está o OpenPosePlugin.
9. Renomeie esta pasta para “Cenário Infinito”.
10. Baixe uma imagem, como a apresentada na Figura A.7, a mesma será utilizada como plano de fundo móvel do jogo.

Figura A.7 – Plano de fundo utilizado para ser o cenário móvel do jogo.

11. Transforme a imagem em um Sprite 2D, clicando em “Texture Type”, na janela “Inspector”, e escolhendo a opção Sprite (2D and UI). Na opção “Wrap Mode” selecione “Repeat”. Depois clique em “Apply” no canto inferior direito, conforme mostra a Figura A.8.

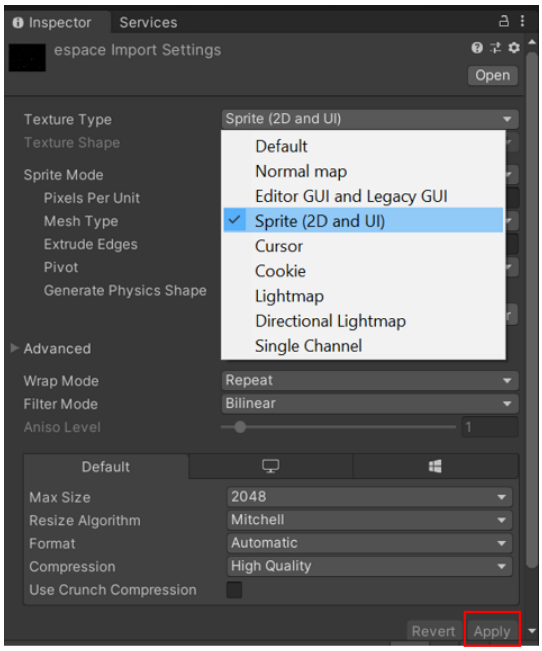

Figura A.8 – Criando um *sprite* 2D.

12. Crie um objeto 3D chamado *Quad* conforme mostra a Figura A.9.

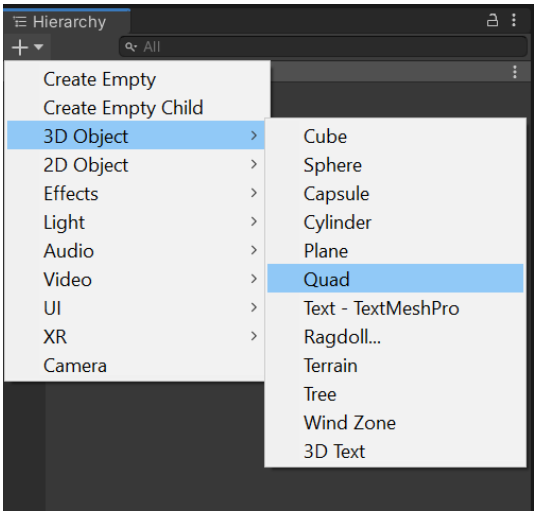

Figura A.9 – Criando objeto 3D “*Quad*”.

13. Resete o *transform* do *Quad* criado, clicando com o botão direito do mouse em cima de seu componente na janela “Inspector” e escolhendo a opção “Reset”.
14. Renomeie o *Quad* criado para “CenarioInfinito”.
15. Adicione a imagem baixada no passo 10, selecionado a mesma na pasta “Cenário Infinito”, criada no passo 9, e arrastando-a sob o *Quad*.

16. Na janela Inspector navegue até o componente "*Shader*" e selecione as opções Unity/Texture, conforme mostra a Figura A.10.

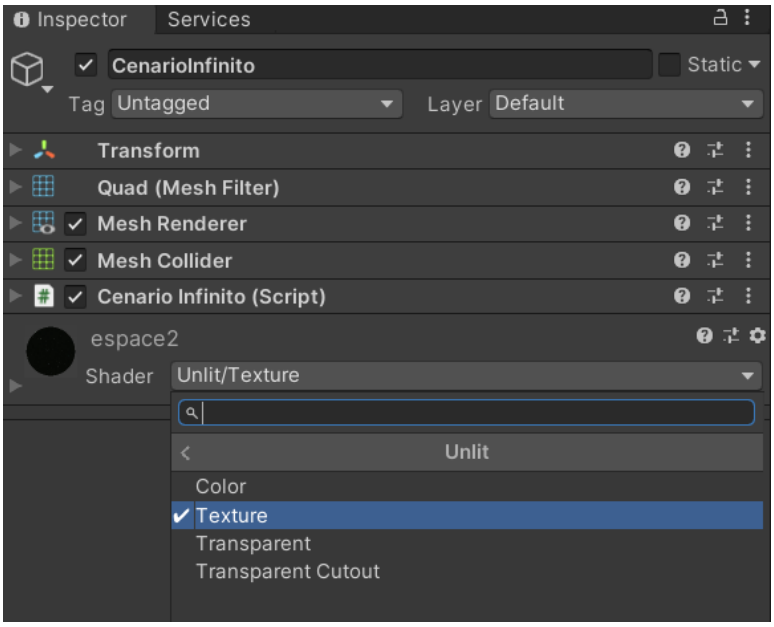

Figura A.10 – Cenário após alterações no componente "*Shader*".

17. Ajuste seu tamanho através da opção "Scale" do componente *transform*. No eixo "x" coloque o valor de 16 e no eixo y coloque o valor 9. A Figura A.11 mostra como ficará o resultado após o ajuste.

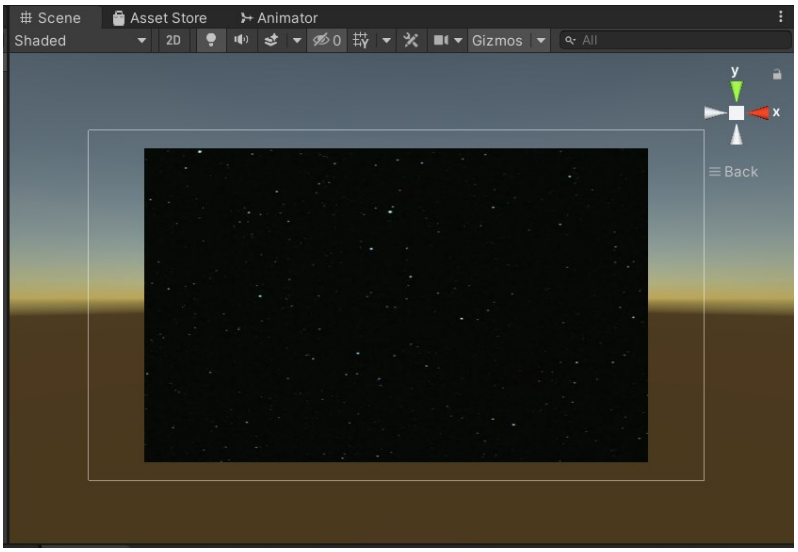

Figura A.11 – Janela "Scene" após ajuste de tamanho do *Quad*.

18. Ajuste o cenário para ficar do mesmo tamanho da área da câmera, que é o retângulo de bordas brancas que contém o cenário. Selecione a ferramenta "Rect tool" e selecione um de seus vértices, com o botão do mouse ainda pressionado, pressione as teclas SHIFT+ALT no teclado e arraste o cenário até ficar do tamanho da área do retângulo mencionado, conforme a Figura A.12.

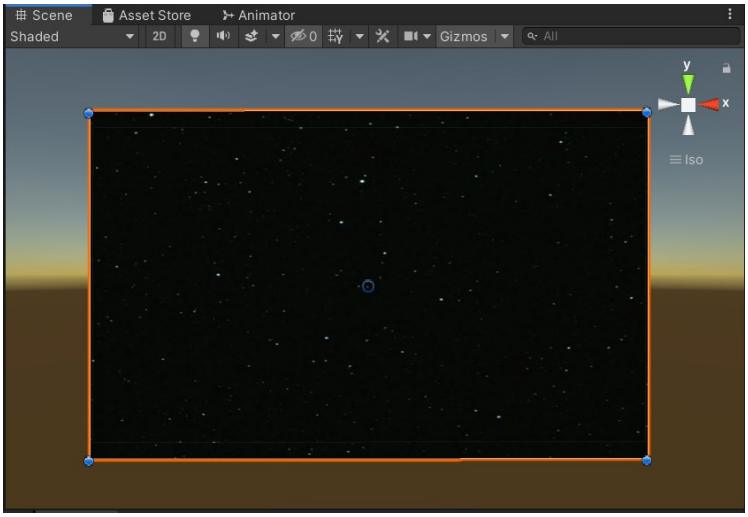

Figura A.12 – Cenário ajustado com a área da câmera.

19. Crie uma pasta chamada “Scripts” na janela “Project”, onde serão colocados todos os scripts utilizados no desenvolvimento do jogo.
20. Dentro da pasta criada, clique com o botão direito do mouse e na opção “Create” clique em “C# Script” para criar o *script* que controlará a movimentação do cenário, conforme mostra a Figura A.13.

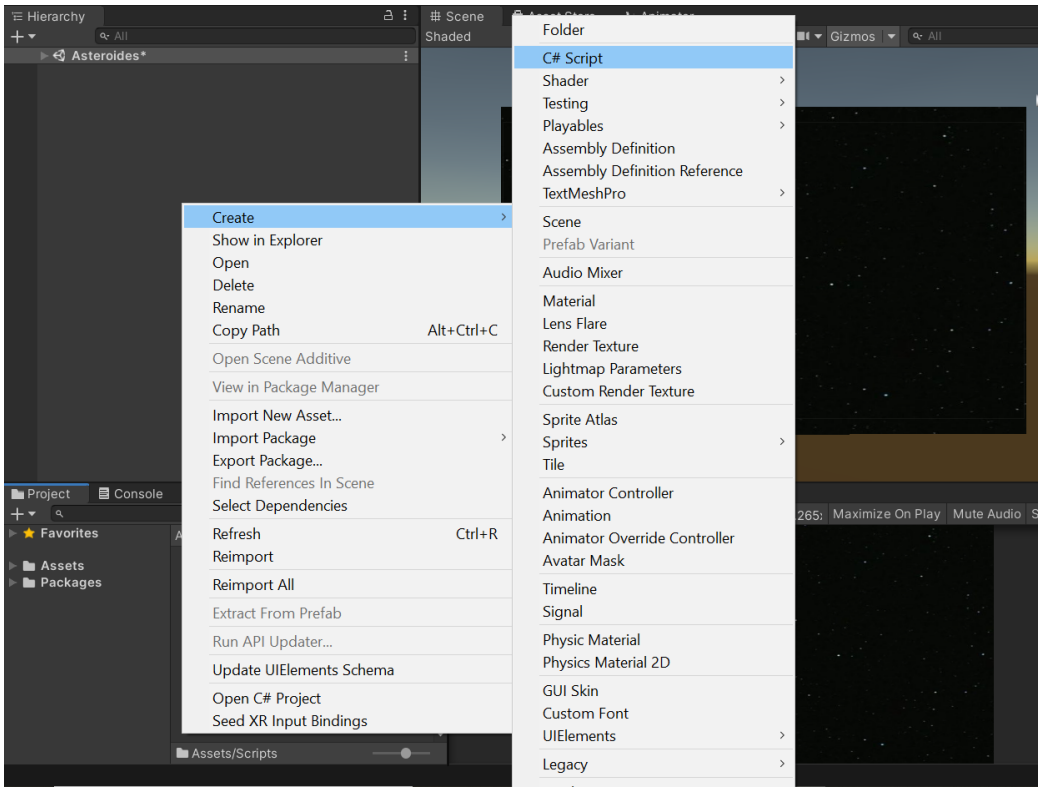

Figura A.13– Criando um novo *script*.

21. Renomeie este *script* criado para “*CenarioInfinito.cs*” e abra-o com seu editor de preferência.

22. Apague todo o conteúdo mostrado na Figura A.14 e copie o conteúdo do script *"CenarioInfinito.cs"*, presente no Apêndice G.

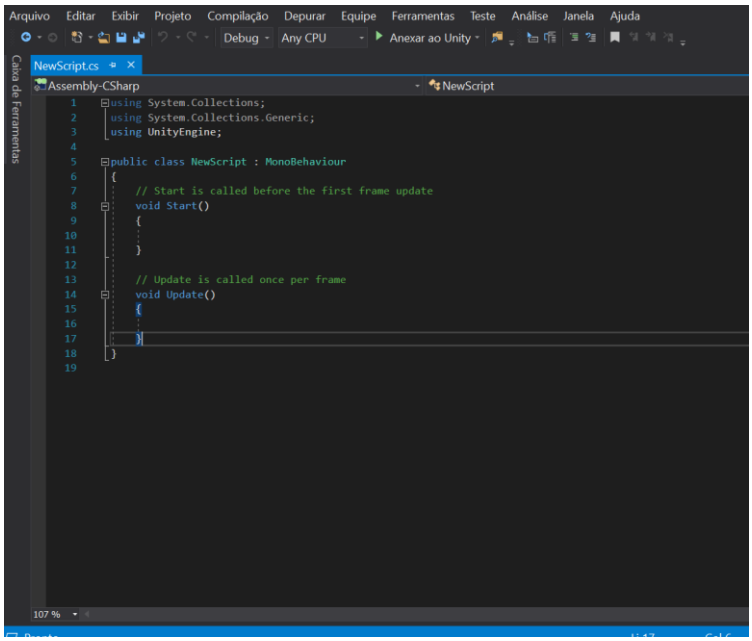

Figura A.14 – Criando o script *"CenarioInfinito.cs"*.

23. Salve o script e volte para a tela do Unity 3D e aguarde alguns segundos até o sinal mostrado na Figura A.15, no canto inferior direito, desaparecer (não é necessário fechar o editor de software).

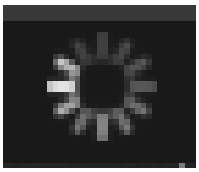

Figura A.15– Sinal de compilação do Script.

24. Adicione o *script* criado para o cenário. Para isto clique e arraste o script para a janela "Inspector" do *Quad* "CenarioInfinito".
25. Atribua o valor 0.03 à variável "Velocidade Scroll", gerada pelo script "CenarioInfinito.cs", como mostra a Figura A.16.

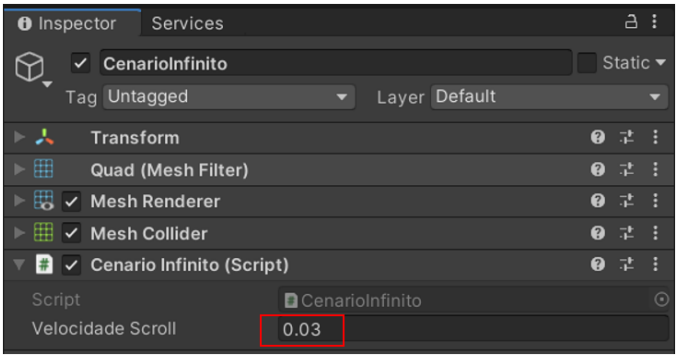

Figura A.16 – Atribuindo a velocidade de movimento ao cenário.

26. No painel “Canvas”, dentro da janela “Hierarchy”, no componente “UI” crie um elemento do tipo “Text”, conforme mostra a Figura A.17.

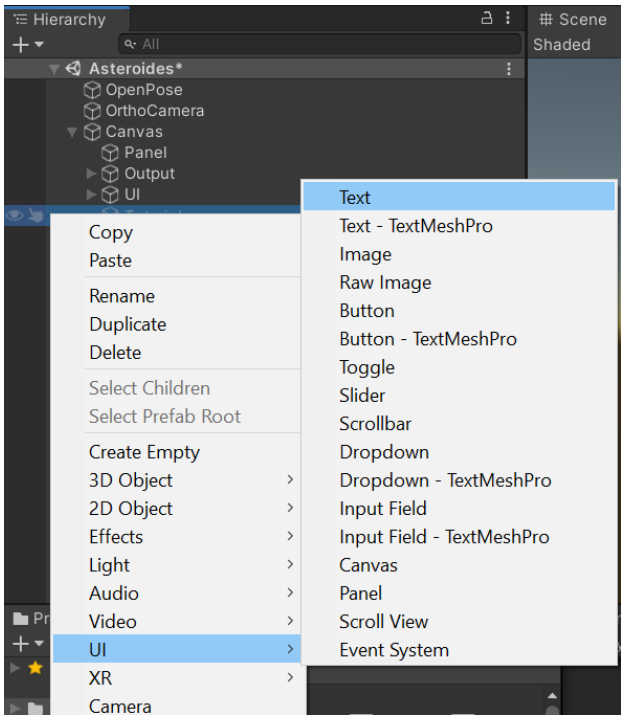

Figura A.17 – Criando elemento de texto.

27. Com este objeto do tipo “Text” selecionado, vá até a janela Inspector e no componente “Text” escreva “Exploda os asteroides”, altere, também, os demais componentes conforme mostra a Figura A.18.

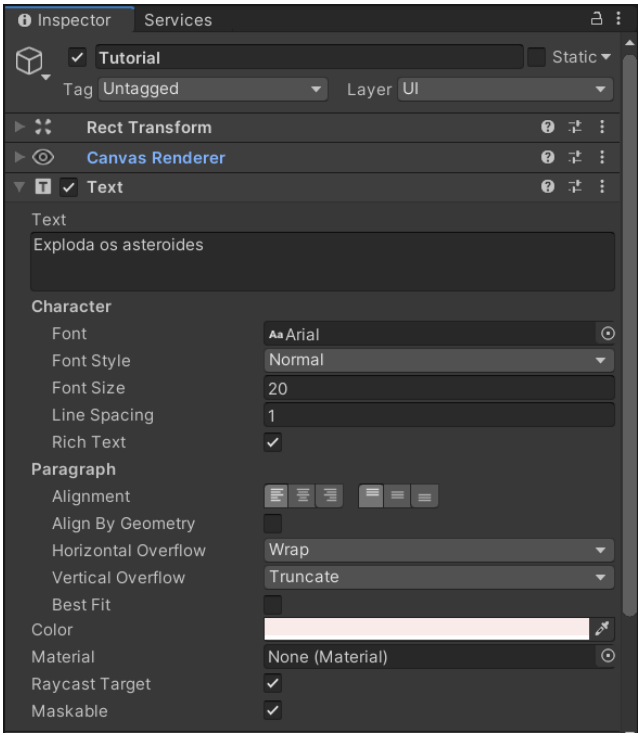

Figura A.18 – Alterando o objeto do tipo “Text” gerado.

28. Posicione o objeto na cena conforme mostra a Figura A.19.

Figura A.19 – Posicionamento do objeto na cena.

29. Baixe gratuitamente o *Asset* “2D Space Kit” na janela “Asset Store” conforme explicado no Apêndice B.
30. Utilizaremos como avatar do jogo o *Prefab* “Cruiser 3”, contido na pasta “Ships and Stations”.
31. Arraste o *Prefab* para a cena do jogo e posicione-o conforme mostra a Figura A.20.

Figura A.20 – Posicionando o avatar do jogo.

32. Para a propulsão da nave baixe o gif “flamme HD”, disponível em:
33. <https://pt.picmix.com/stamp/flamme-HD-1063045>
34. Converta o gif em sprites seguindo os passos contidos no Apêndice C.
35. Crie uma pasta dentro de “Assets” na janela “Project”, renomeie para “Animações” e coloque o arquivo gerado na conversão dentro dela, com nome “chama”.

36. Com o arquivo ainda selecionado, vá na janela “Inspector” e em “Texture Type” escolha a opção “Sprite (2D and UI)”, em “Sprite Mode” escolha “Multiple”, clique no botão “Sprite Editor” e na caixa de diálogo que aparecer clique em “Apply”. No canto superior esquerdo clique em “Slice” e em “Type” selecione “Grid by Cell Count”, em “column e Row” atribua o valor 5 para “C” e 1 para “R”, clique em “Apply” no canto superior direito e feche a janela.
37. De volta à janela “Inspector” vá até o componente “Compression” e escolha “High Quality”, clique no botão Apply.
38. Arraste o arquivo “chama.png” para a tela do jogo, na caixa de diálogo que surgir selecione a pasta “Animações”, caso ainda não estiver selecionada, e salve o arquivo como “chamamov.anim”.
39. Posicione o arquivo conforme mostra a Figura A.21.

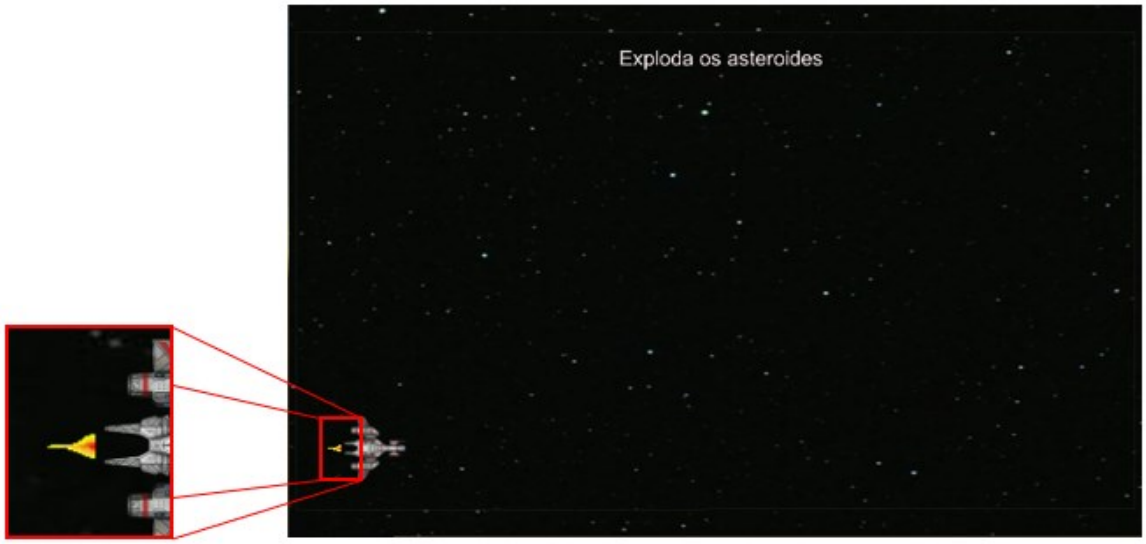

Figura A.21 – Posicionando a animação da chama propulsora da nave.

40. Na janela “Hierarchy” renomeie o nome do arquivo para “Propulsão” e coloque-o como filho do objeto “Cruiser”, conforme mostra a Figura A.22.

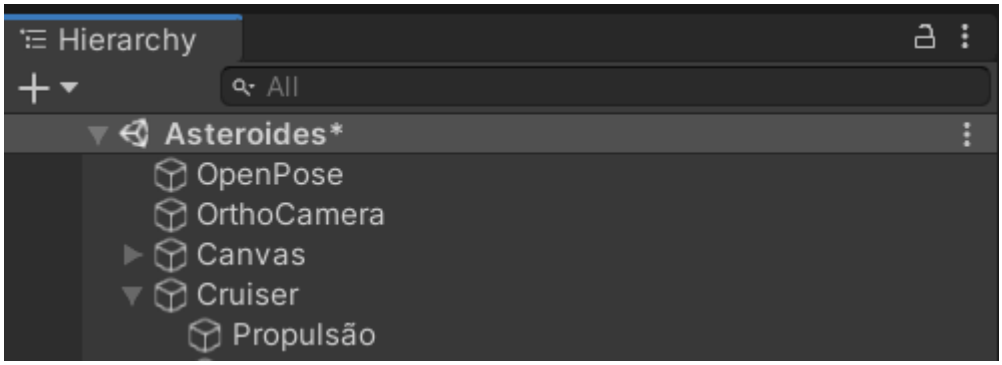

Figura A.22 – Colocando o objeto gerado como sendo filho do objeto “Cruiser”.

41. Crie um outro objeto filho do objeto “Cruiser”, desta vez um objeto vazio, clicando com o botão direito do mouse em cima de “Cruiser” e escolhendo a opção “Create Empty”, como mostra a Figura A.23.

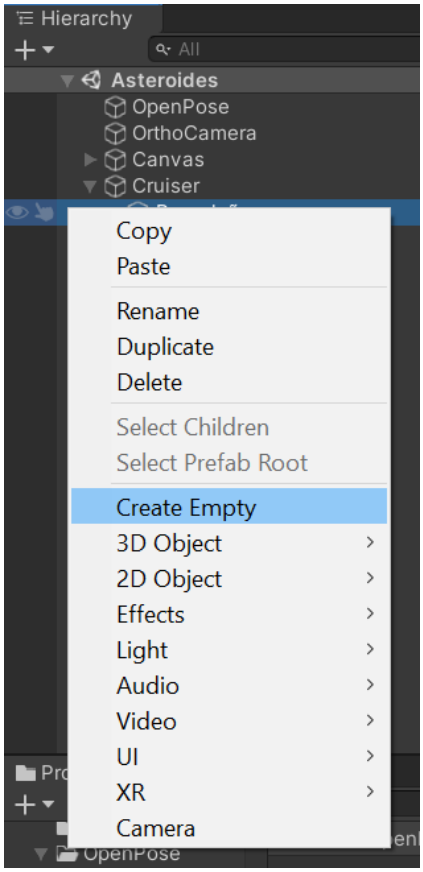

Figura A.23 – Criando um objeto vazio.

42. Renomeio o objeto criado para “FirePoint”.
43. Posicione o objeto a frente da nave como demonstrado na Figura A.24.

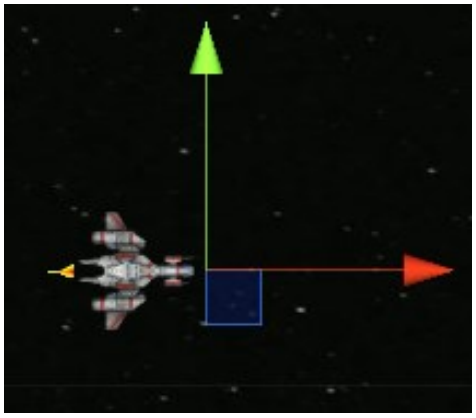

Figura A.24 – Posicionando o objeto “FirePoint”.

44. Com o objeto “FirePont” selecionado, vá até a janela “Inspector” e crie um novo *script* através do botão “Add Component”, conforme mostra a Figura A.25.

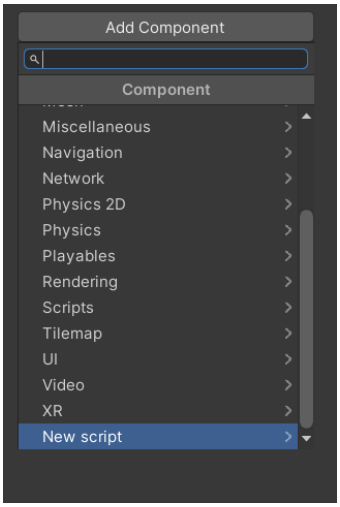

Figura A.25 – Criando um novo script através do botão “Add Component” da janela “Inspector”.

45. Renomeie o script criado para “*spawnbullet.cs*” e abra-o.
46. Apague o conteúdo contido nele e copie o código do Apêndice I.
47. Salve e volte para a Unity.
48. Vá até a pasta “2D Space Kit”, baixada no passo 29, e dentro dela abra a pasta “Prefabs”. Selecione o objeto “Projectile Sharp.prefab” e renomeie o mesmo para “Bullet.prefab”. De volta a janela “Inspector” do objeto “FirePoint”, no novo componente “Spawnbulett”, vá até “Spawn” e na caixa de seleção escolha o *Prefab* “Bullet”, conforme é demonstrado na Figura A.26.

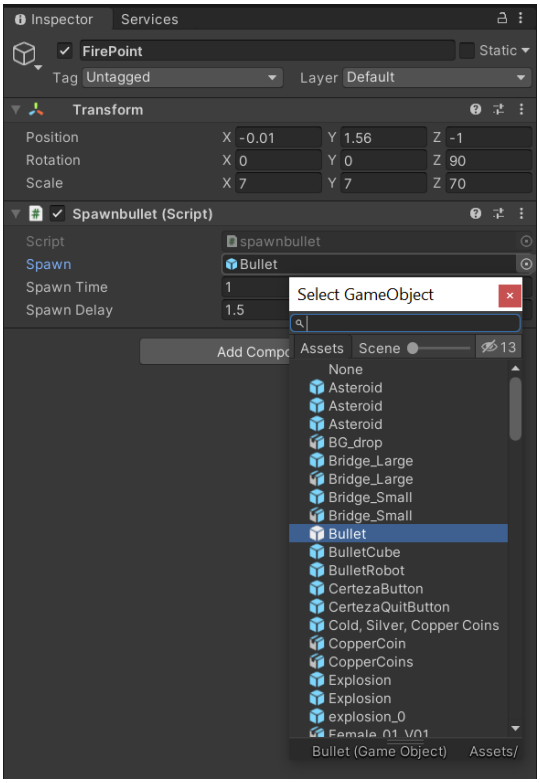

Figura A.26 – Criando o *spawn* do *Prefab* “Bullet”.

49. Em “Spawn Time” coloque “1” e em “Spawn Dellay” coloque “1.5”, conforme a Figura A.26.
50. Abra o Prefab “Bullet” clicando duas vezes no mesmo.
51. Crie um novo *script* seguindo o passo 42 e renomeie ele para “*Bullet.cs*”.
52. Abra o script criado, apague o conteúdo do mesmo e copie o código do apêndice J.
53. Salve as alterações e retorne ao Unity.
54. Ainda com o *Prefab* “Bullet” aberto, vá na janela “Inspector” e adicione os componentes “Rigidbody 2D” e “Capsule Collider 2D”, ambos contidos no botão “Add Component” na opção “Physics 2D”, e altere seus valores conforme a Figura A.27.

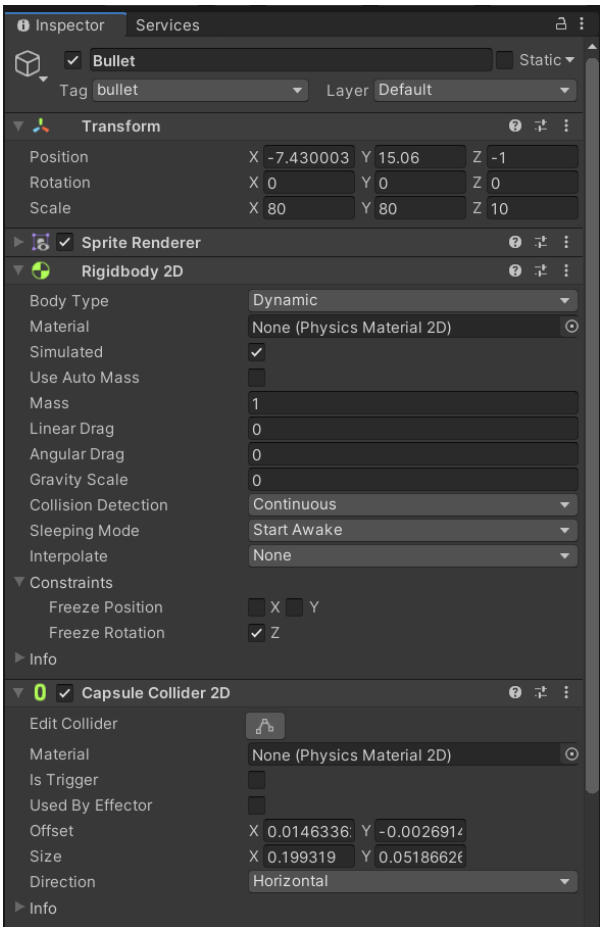

Figura A.27 – Adicionando os componentes “Rigidbody 2D” e “Capsule Collider 2D” ao *Prefab* “Bullet”.

55. Ajuste o “Capsule Collider 2D” ao *Prefab* “Bullet” conforme indicado na Figura A.28.

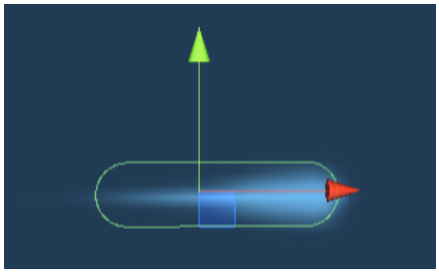

Figura A.25 – Ajuste do “Capsule Collider 2D” ao *Prefab* “Bullet”.

56. Vá até o componente “Speed”, adicionado através do script “*Bullet.cs*”, e coloque o valor “200”, depois vá até o componente “Rb”, logo abaixo, e arraste o componente “RigidBody 2D” até o mesmo, conforme mostra a Figura A.29.

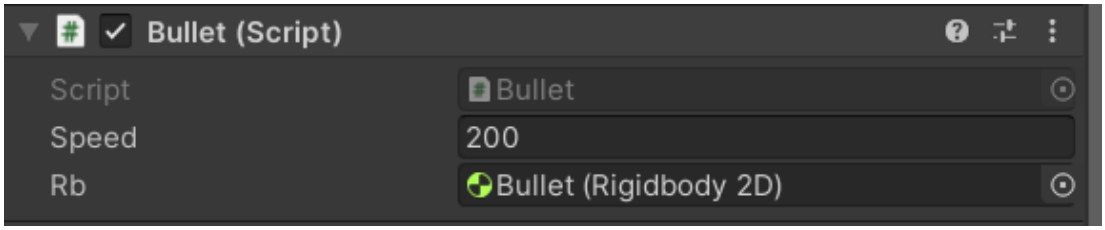

Figura A.26 – Adicionando os elementos necessários para o funcionamento do *script* “*Bullet.cs*”.

57. Na janela “Hierarchy”, crie um objeto vazio como explicado no passo 39.
58. Renomeie o objeto criado para “Asteroid”.
59. Posicione o objeto no ponto de intersecção dos eixos x e y, conforme indicado na Figura A.30.

Figura A.30 – Posicionando o objeto “Asteroid” na tela de jogo.

60. Já na janela “Inspector”, crie um novo *script* seguindo o passo 42.
61. Renomeie o script criado para “*spawnAsteroid.cs*” e abra-o.
62. Apague o conteúdo do arquivo e copie o código do Apêndice K.
63. Salve as alterações e retorne ao Unity.
64. Siga o passo 26 e crie um elemento do tipo “Text”.
65. Renomeie o elemento criado para “Contador_asteroides”.
66. Posicione o elemento no ponto de intersecção dos eixos x e y, conforme mostra a Figura A.31.

Figura A.31 – Posicionando o elemento “Contador_asteroides” na tela de jogo.

67. Vá novamente até a pasta “2D Space Kit”, baixada no passo 29. Dentro dela abra a pasta “Asteroids”, selecione o objeto “Asteroid 5” e arraste para a janela “Hierarchy”.
68. Com o objeto ainda selecionado, arraste-o para a pasta “Prefabs”, dentro da pasta “2D Space Kit”. O objeto se tornará um *Prefab*.
69. Renomeie o objeto para “*Asteroid.prefab*”.
70. Crie uma animação para o objeto, aproximando-o do avatar. Para isso, na janela “Inspector” clique em “Add Component” e na barra de pesquisa que irá surgir digite “Animator” e selecione-o.
71. Habilite a janela “Animation”, caso ainda não esteja habilitada. Para isso vá até a barra de ferramentas e no menu “Window” escolha a opção Windows/Animation, como demonstrado na Figura A.32.

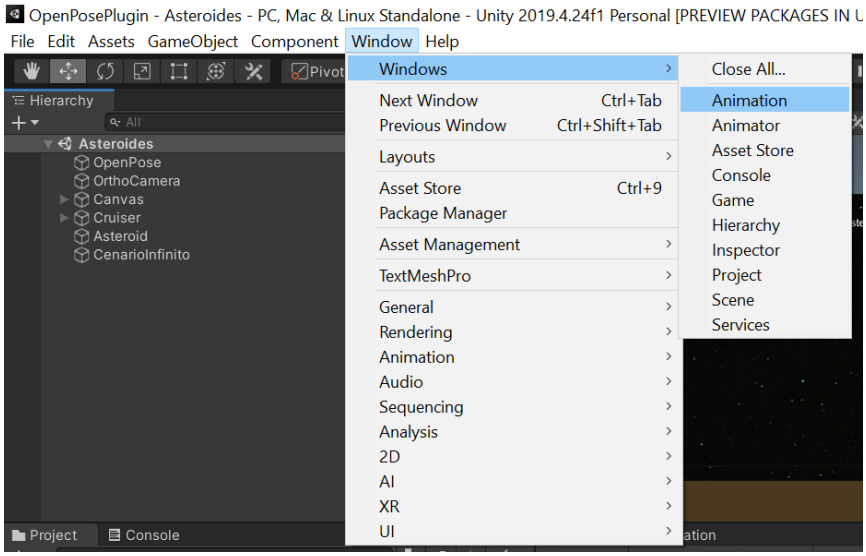

Figura A.32 – Habilitando a janela “Animation”.

72. Com o objeto selecionado vá na janela “Animation” e clique em “*Create*”, como mostra a Figura A.33.

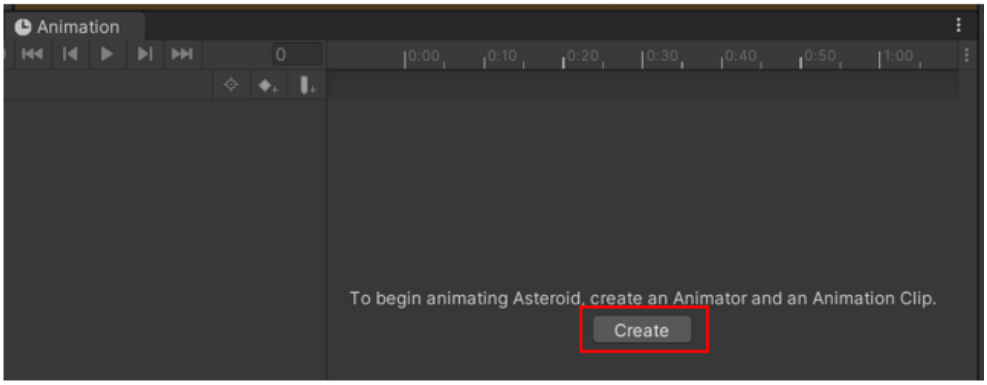

Figura A.33 – Criando uma nova animação.

73. Na janela que irá surgir salve a animação como “*AsteroidAnimation.anim*” na pasta “Animações”, criada no passo 34.
74. Clique no Botão indicado na Figura A.34 para iniciar a gravação da animação.

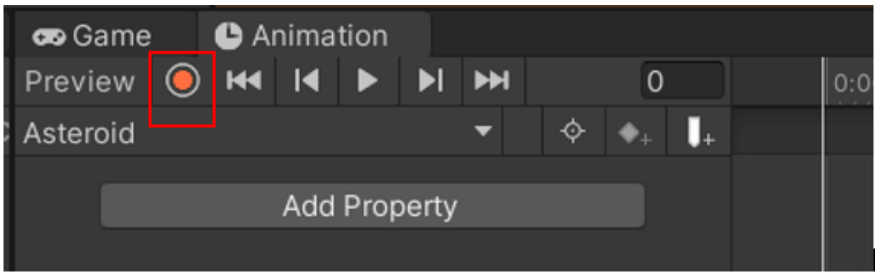

Figura A.34 – Iniciando a gravação da animação.

75. Com a linha branca no índice “0:00”, posicione o objeto na tela de jogo como mostra a Figura A.35. Coordenadas: x: 180,9; y: -71,8; z: -2.

Figura A.35 – Posição no tempo “0:00” da animação.

76. Arraste a linha branca do marcador para o índice “3:00” e posicione o objeto como indicado na Figura A.36. Coordenadas: x: 100,6; y: -71,8; z: -2.

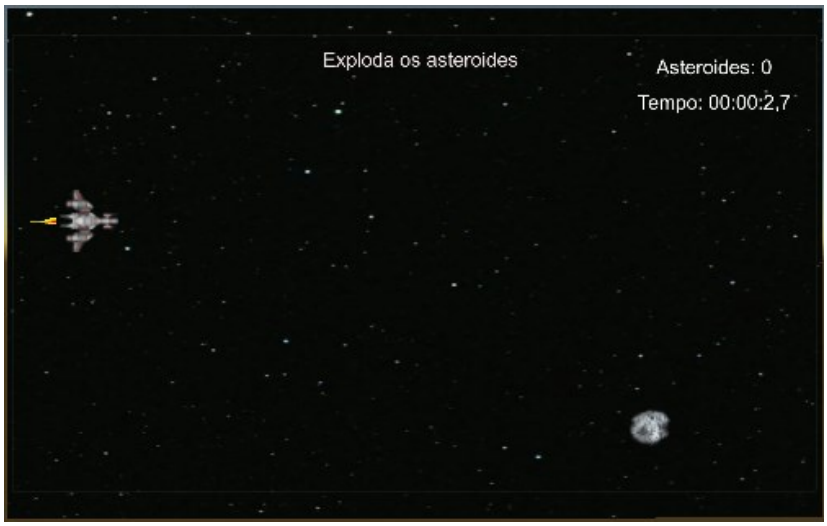

Figura A.36 – Posição no tempo “3:00” da animação.

77. Arraste a linha branca marcador para o índice “40:16” e mantenha o objeto posicionado como indicado na Figura A.36. Coordenadas: x: 100,6; y: -71,8; z: -2.
78. Clique novamente no botão indicado na Figura A.34 para parar a gravação e finalizar a animação.
79. Exclua o Prefab “Asteroid” da janela “Hierarchy”, apenas desta janela, ele ficará na pasta “Prefabs”.
80. Na janela “Inspector” do objeto vazio “Asteroid”, vá até o componente “Spawnee” e arraste para ele o *Prefab* “Asteroid”.
81. Em “Spawn Time” coloque o valor “0” e em “Spawn Delay” coloque o valor “5”, conforme mostra a Figura A.37.

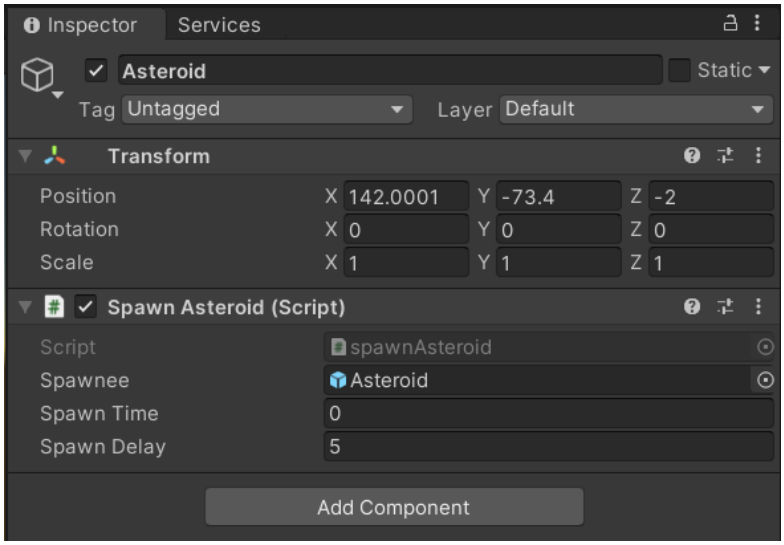

Figura A.37 - Adicionando os elementos necessários para o funcionamento do *script* “*spawnAsteroid.cs*”.

82. Crie mais dois objetos vazios e renomeie eles para “Asteroid2” e “Asteroid3”, posicione os mesmos na tela, acima do objeto citado no passo 58.
83. Repita os passos 66 a 79 criando outros dois locais de *spawn* de asteroides, modificando a coordenada “y” de cada um para “-23” e “18.6”, respectivamente. Altere também o valor “Spawn Time” e “Spawn Delay” de acordo com o tempo desejado para os asteroides surgirem após o primeiro ciclo terminar.
84. Baixe a imagem, com sprites de uma explosão, disponível gratuitamente no link abaixo: <https://www.kissclipart.com/top-down-explosion-sprite-clipart-sprite-clip-art-c1wfi3/>
85. A Figura A.38 mostra como é a imagem a ser baixada.

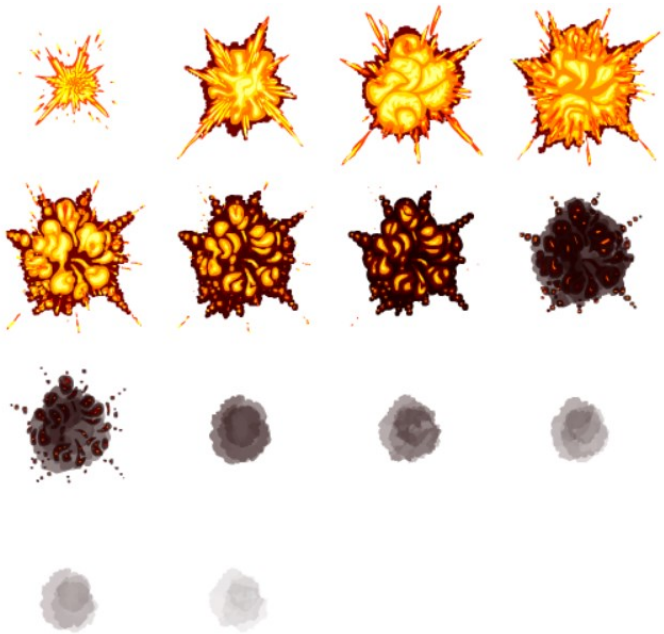

Figura A.38 – Imagem com os sprite de explosão.

86. Coloque a imagem dentro da pasta “Animações”, criada no passo 34.
87. Renomeie a imagem para “explosion”.
88. Transforme a imagem em um *Sprite* 2D, seguindo o passo 11.
89. Arraste o sprite criando para a janela “Hierarchy”, criando um objeto de jogo.
90. Uma nova janela irá abrir e será solicitado um nome para salvar o arquivo gerado, este será a animação de explosão. Coloque o nome “explosion_0”.
91. Salve a animação na pasta “Animações”, criada no passo 34.
92. Juntamente foi criado um Prefab de nome “explosion_0”.
93. Durante a execução do jogo, quando um *Prefab* “Bullet” atinge um *Prefab* “Asteroid”, este último é destruído e surge em seu lugar a animação de uma explosão. Para isso abra o Prefab “Asteroid” e faça as modificações do passo à seguir.

94. Crie um novo *script* na janela “Inspector”, seguindo o passo 42.
95. Renomeie o *script* criado para “*ast.cs*”.
96. Abra o *script*, apague seu conteúdo e copie o código contido no Apêndice L.
97. Salve as alterações e retorne ao Unity.
98. Ainda na janela “Inspector”, adicione os componentes “Circle Collider 2D” e “Rigidbody 2D”, seguindo o passo 53, e configure-os conforme mostra a Figura A.39.

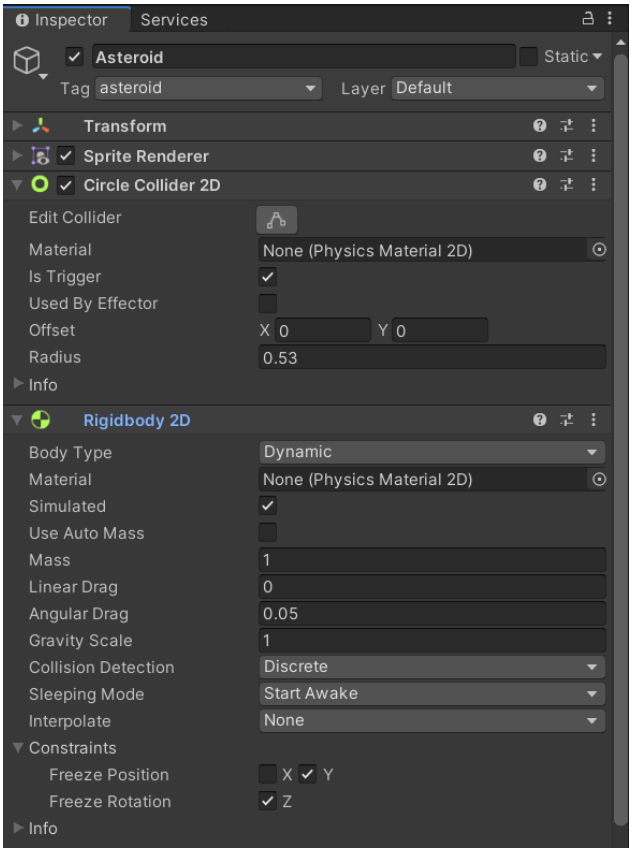

Figura A.39 - Adicionando os componentes “Circle Collider 2D” e “Rigidbody 2D” ao *Prefab* “Asteroid”.

99. Posicione o “Circle Cllider 2D” confore mostrado na Figura A.40.

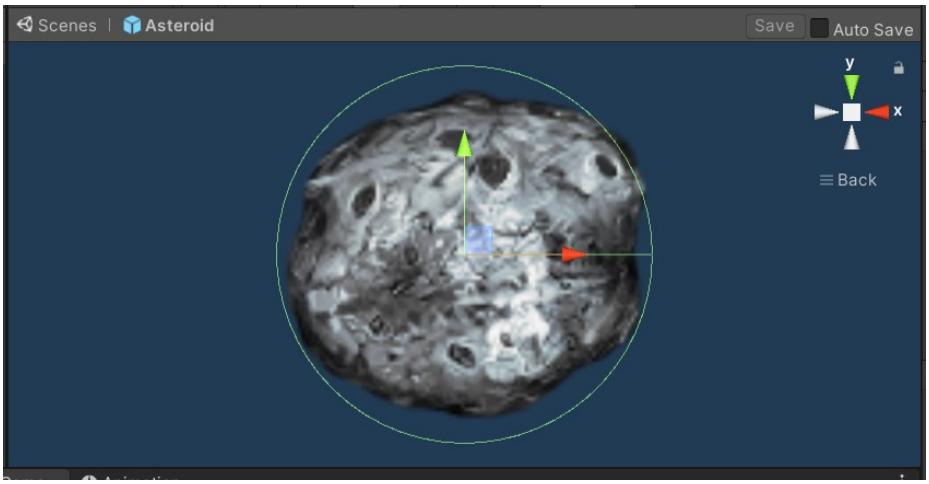

Figura A.40 – Posicionando o “Circle Collider 2D”.

100. Vá até o componente “Explosão”, gerado pelo *script* “*ast.cs*”, e selecione o Prefab “explosion_0”, criado nos passos 74 a 78, conforme mostra a Figura A.41.

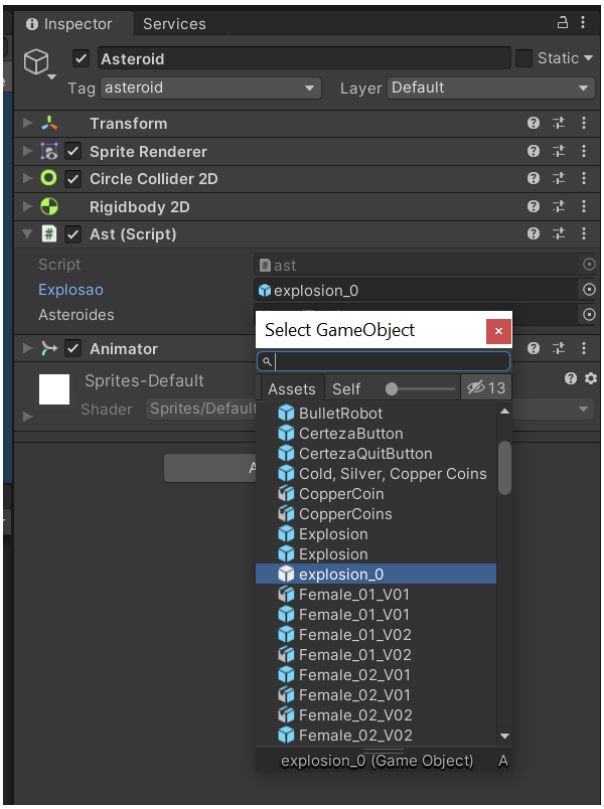

Figura A.41– Adicionando o componente para o funcionamento do *script* “*ast.cs*”.

101. Volte à tela do jogo e salve as alterações clicando na seta indicada na Figura A.42.

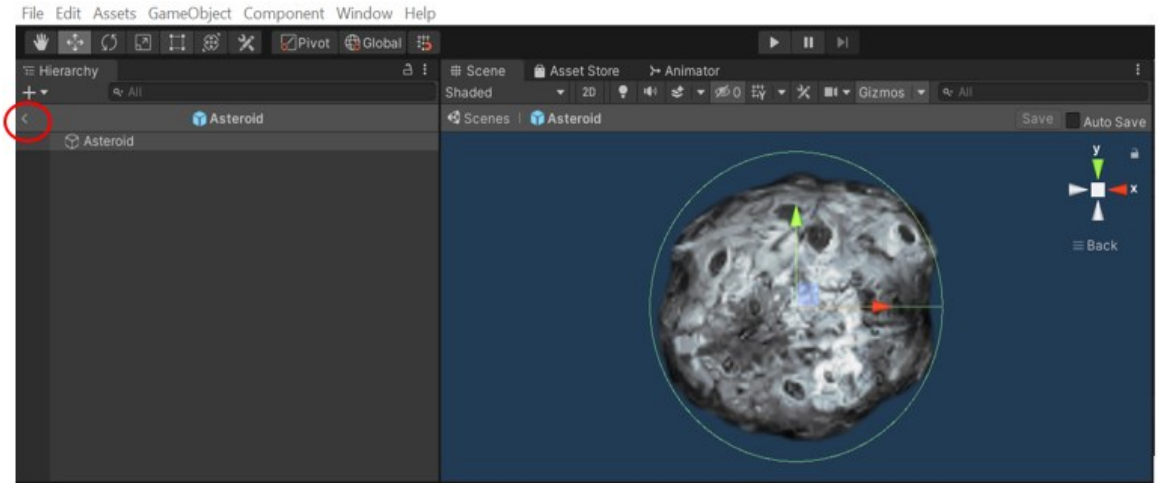

Figura A.42 – Voltando à tela de jogo e salvando as alterações no Prefab “Asteroid”.

102. Crie um objeto do tipo “Text” no painel “Canvas”, seguindo o passo 26.

103. Renomeie o objeto para “Tempo”.

104. Na janela “Inspector”, crie um novo script seguindo o passo 42.

105. Abra o script criado, apague o conteúdo e copie o código “*Tempo.cs*”, contido no Apêndice M.

106. Salve as alterações no documento e volte ao Unity.
107. Ainda na janela “Inspector”, vá até o componente “Execution” e arraste para ele o objeto “Tempo”, contido no painel “Canvas”.
108. Na janela “Hierarchy”, clique no objeto “OpenPose” e, na janela “Inspector, altere os valores dos componentes para que fiquem de acordo com a Figura A.43.

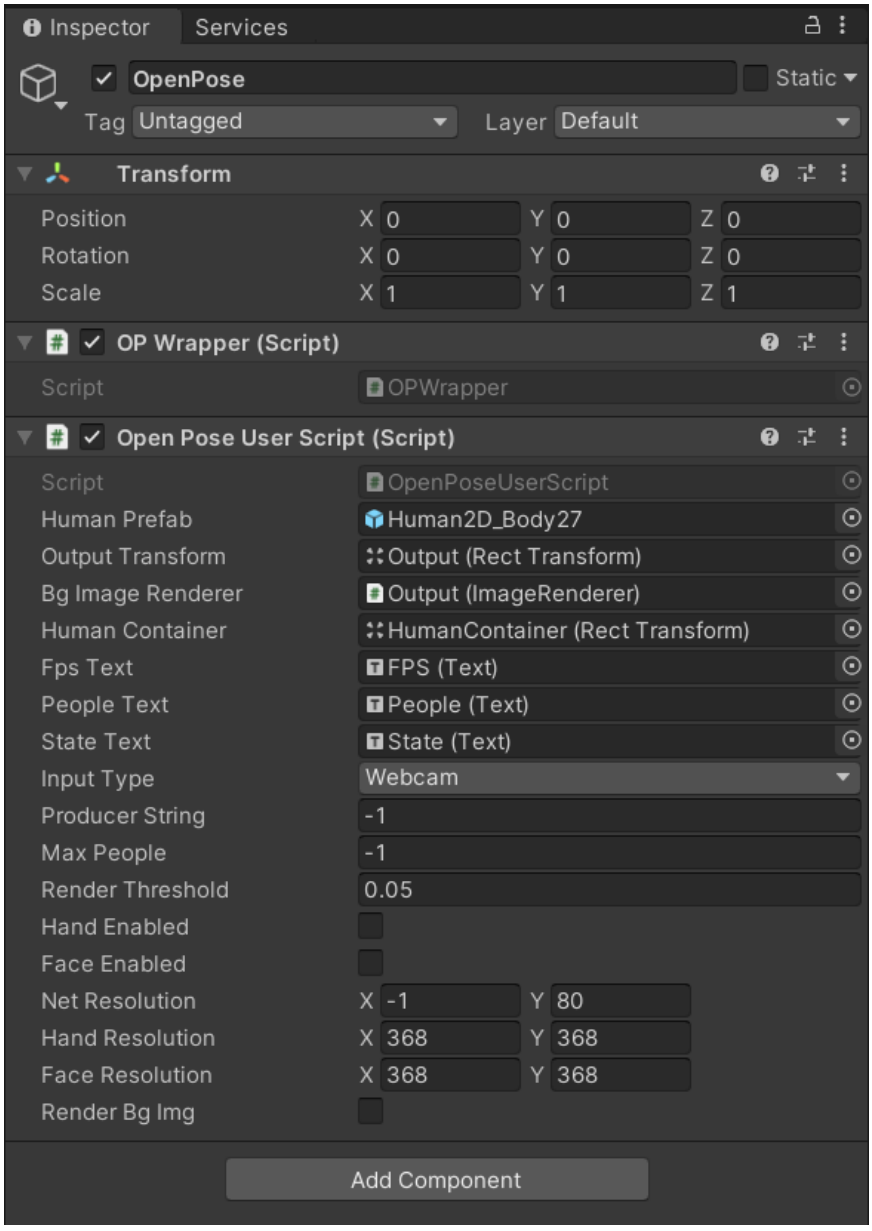

Figura A.43 – Configurando a janela “Inspector” do objeto “OpenPose”.

109. Na janela “Hierarchy”, crie um objeto de jogo vazio, como no passo 39.
110. Renomeie o objeto criado para “Music”.
111. Com o objeto ainda selecionado, vá até a janela “Inspector” e adicione um componente de áudio, clicando no botão “Add Component” e depois escolhendo a opção Audio/Audio Source, como mostra a Figura A.44.

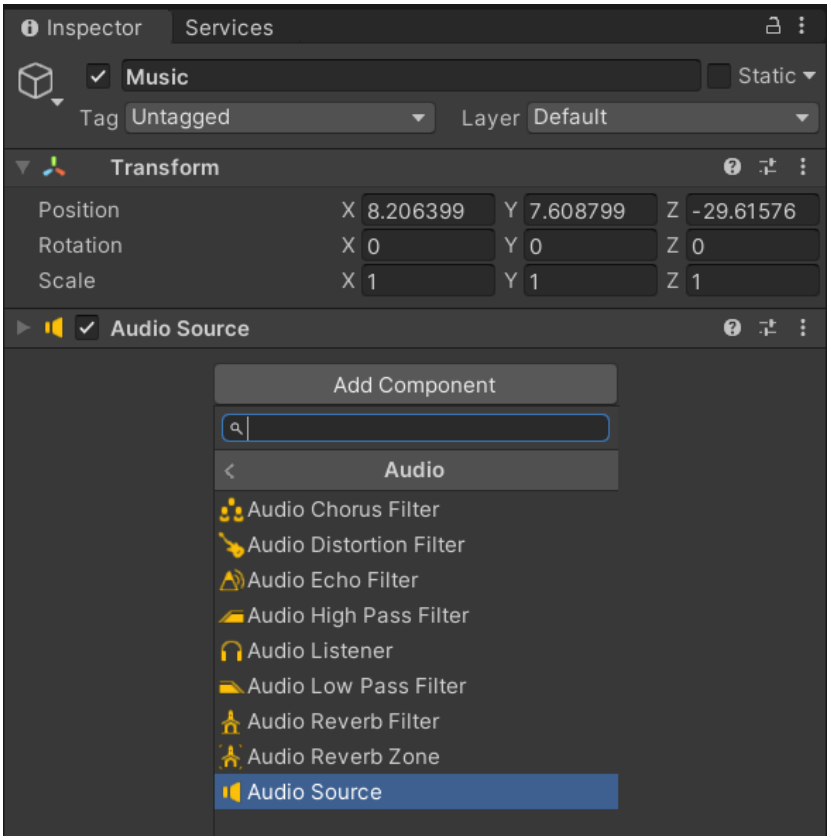

Figura A.44 – Adicionando um componente de áudio no jogo.

112. Baixe gratuitamente o *asset* “Sci-Fi Soundtrack”, ilustrado na Figura A.45, através da janela “Asset Store”.

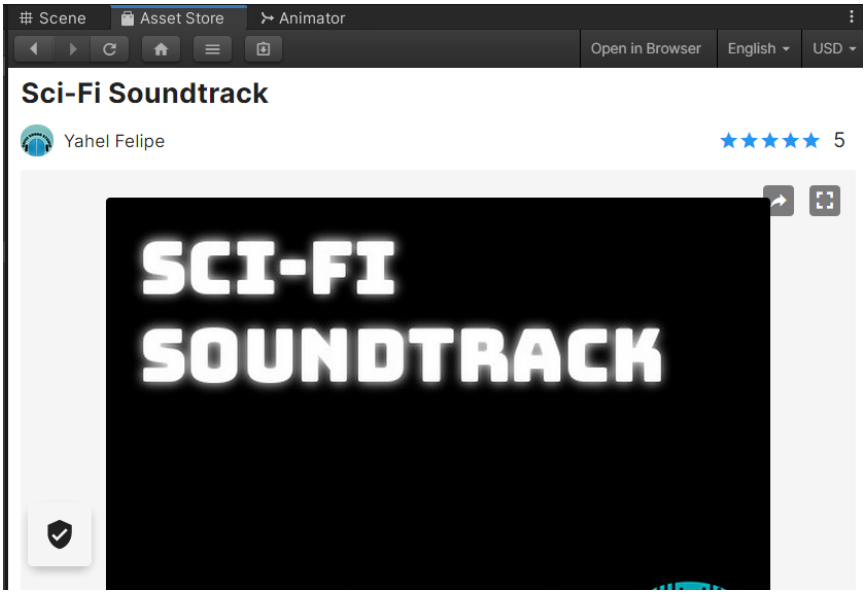

Figura A.45 – Baixando o *asset* “Sci-Fi Soundtrack”.

113. Abra a pasta baixada e arraste a música “First Contact.mp3” para o componente “Audio Source”, da janela “Inspector” referente ao objeto “Music”, e configure tal componente como mostra a Figura A.46.

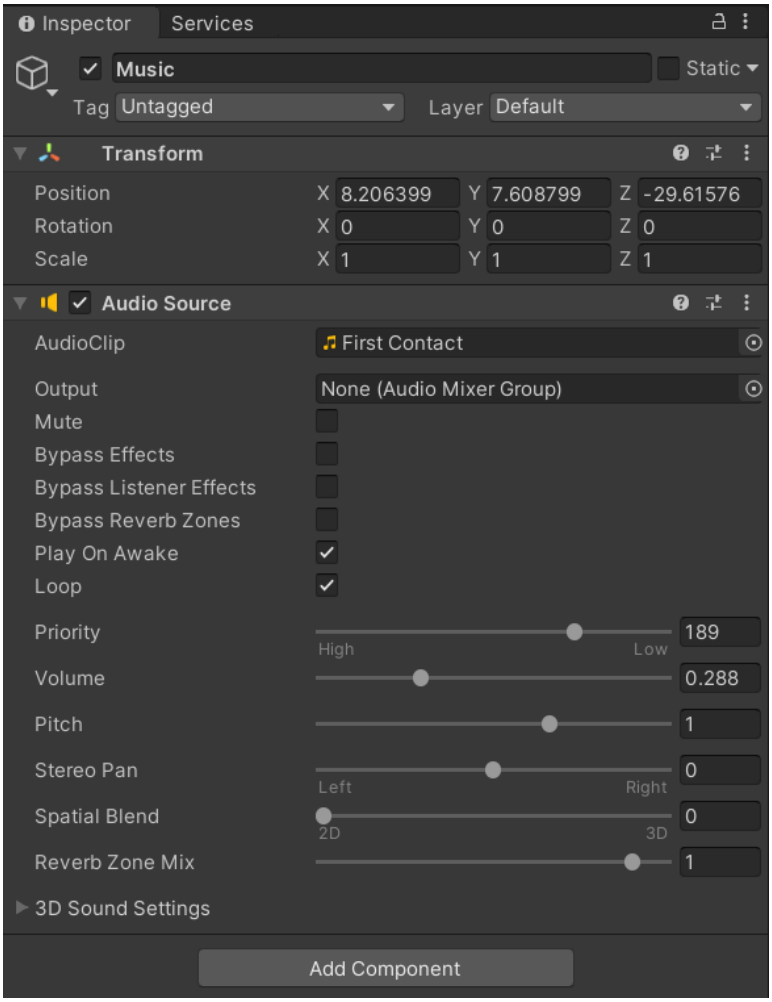

Figura A.46 – Configurando o componente “Audio Source” do objeto “Music”.

114. Baixe gratuitamente o asset “Free Sound Effects Pack”, ilustrado na Figura A.47.

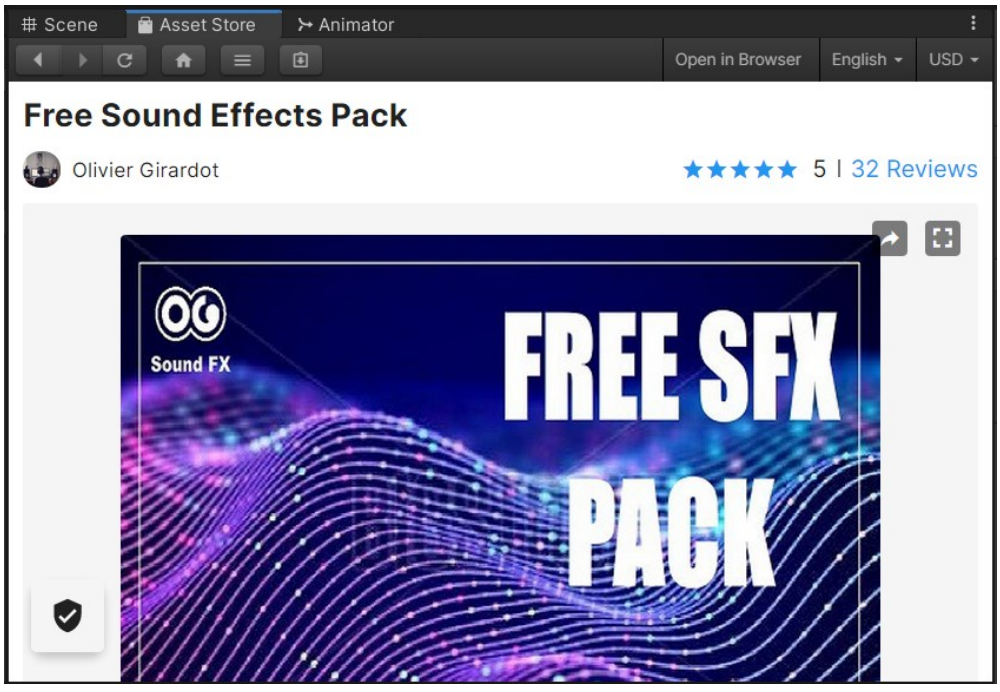

Figura A.47 – Baixando o *asset* “Free Sound Effects Pack”.

115. Abra o *Prefab* “Bullet” e na janela “Inspector” adicione um componente de áudio, como no passo 107.
116. Arraste para o componente “Audio Source” o áudio “Bullet Impact 14.mp3”, contida na pasta “Free Pack”.
117. Configure o componente como o descrito na Figura A.48.

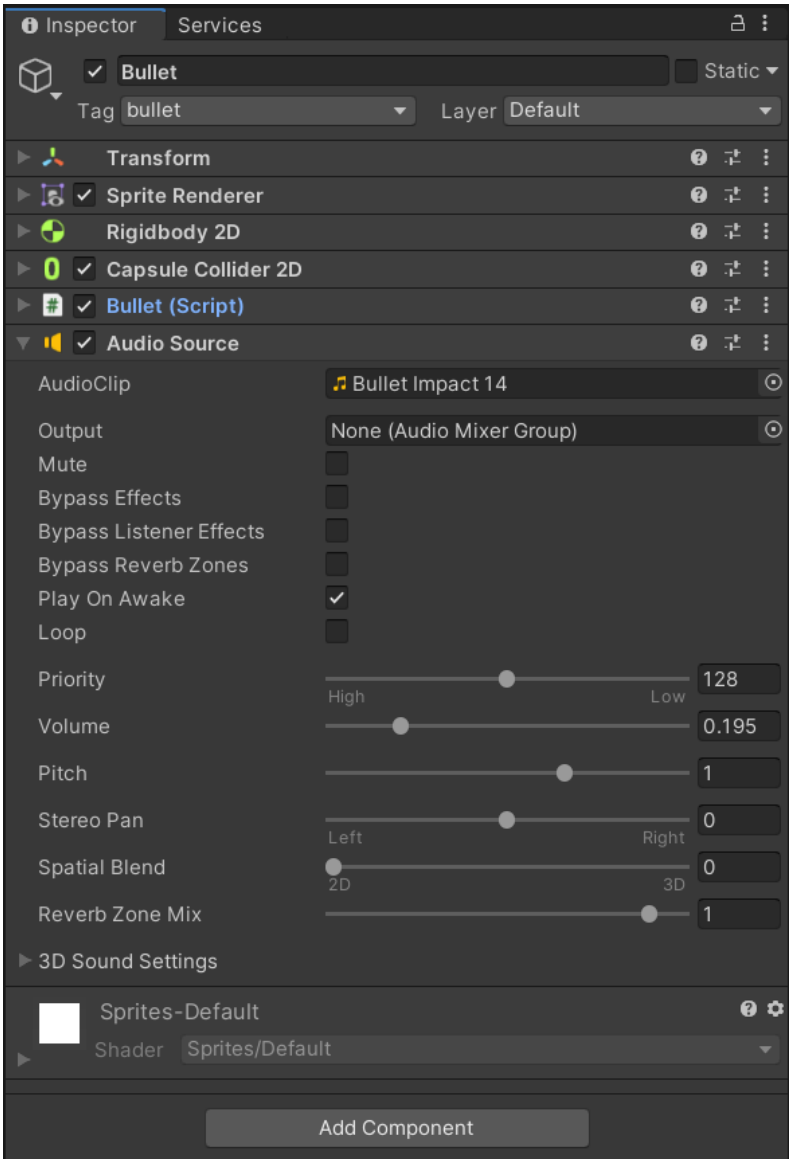

Figura A.48 – Configurando o componente “Audio Source” do *Prefab* “Bullet”.

118. Salve as alterações.
119. Abra o Prefab “explosion_0” e na janela “Inspector” adicione um componente de áudio, como no passo 107.
120. Arraste para o componente “Audio Source” o áudio “Explosion 1.mp3”, contida na pasta “Free Pack”.
121. Configure o componente como o descrito na Figura A.49.

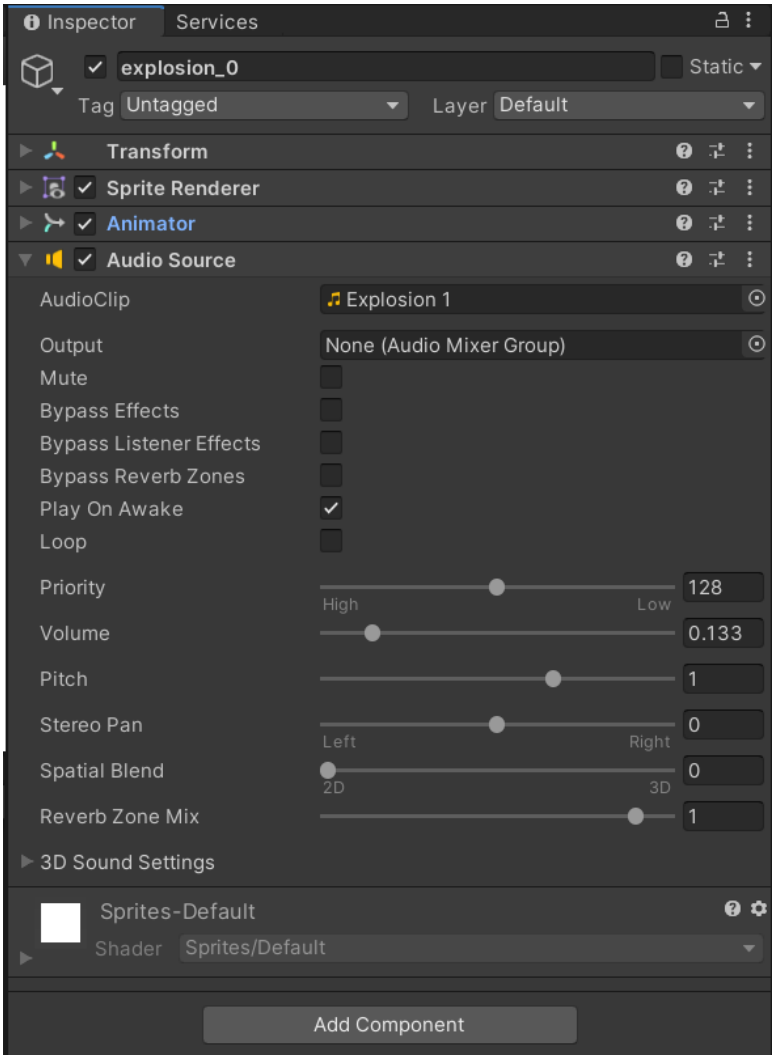

Figura A.49 - Configurando o componente “Audio Source” do *Prefab* “explosion_0”.

122. Salve as alterações.

Desta forma, finaliza-se o tutorial de um dos jogos desenvolvidos neste trabalho. Durante a escrita, alguns pontos foram simplificados ou não detalhados devido a serem questões estéticas ou de configurações flexíveis que foram implementadas apenas a título de curiosidade e conhecimento mais amplo do software.

Para se executar o jogo, desde que o mesmo não possua procedimentos em aberto (como, por exemplo, componentes a serem completados com variáveis), basta pressionar o botão “Play”. Pausa-se a execução ou encerra-se a mesma, pressionando os botões específicos para cada uma destas ações.

Todos estes botões citados estão destacados na Figura A.50.

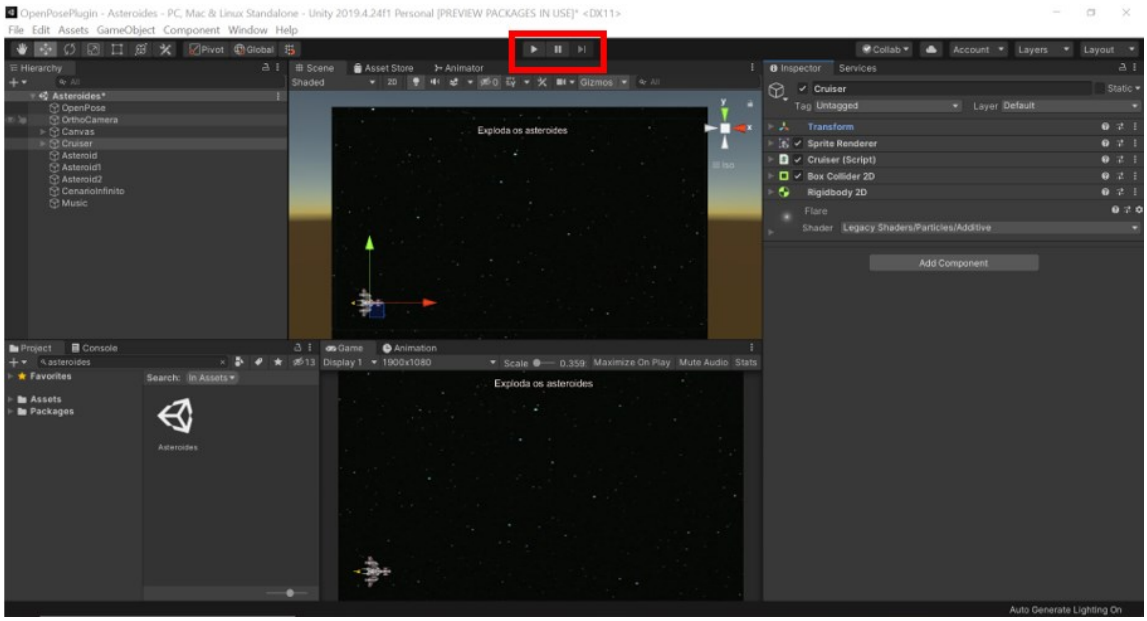

Figura A.50 – Destaque aos controles de execução do jogo.

Após a finalização, é possível se criar um executável do jogo desenvolvido no Unity 3D. Para isto, navegue até o menu "*File*" e então clique na opção "*Build Settings...*" (Ctrl+Shift+B) e, na janela que irá aparecer selecione a plataforma desejada, assim como apresentado na Figura A.51. Para este projeto, desenvolveu-se um jogo para Windows com uma arquitetura tanto de 32 bits quanto de 64 bits. A criação deste executável não necessita de nenhum método de compressão, uma vez que o jogo não possuí uma complexidade extremamente alta. Sendo assim, deve-se clicar em "*Build*" e aguardar a criação do arquivo executável.

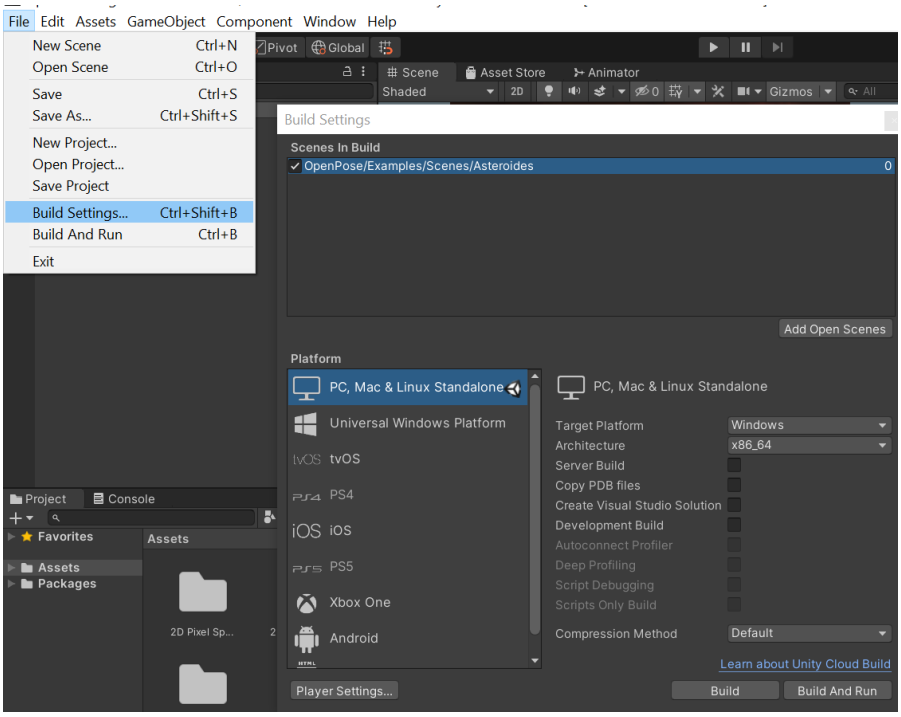

Figura A.51 – Janela de criação de um arquivo executável para o jogo desenvolvido.

B – TUTORIAL PARA BAIXAR *ASSETS* – "*ASSET STORE*"

Para este tutorial será utilizado o *asset* "*2D Space Kit*", utilizado durante a criação do jogo "Asteroides" e citado no passo 29 do tutorial de desenvolvimento do mesmo. Este *asset* é disponibilizado gratuitamente por *Brett Gregory*.

Para baixá-lo acesse a *Asset Store* da Engine Unity 3D e faça login em sua conta. No mecanismo de busca da própria loja, procure pelo termo "2D Space Kit" (Disponível em : https://assetstore.unity.com/packages/2d/environments/2d-space-kit-27662 - Acesso em 13/10/2021). Clique em "*Add to My Assets*", como mostra a Figura B.1.

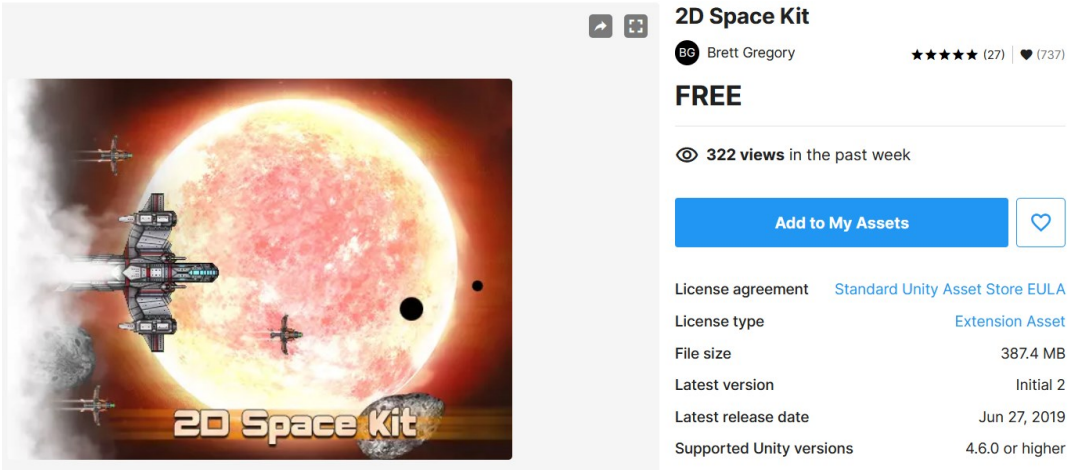

Figura B.1 – Visualização de parte da tela de aquisição do *asset* "*2D Space Kit*" no *Browser*.

Através desta ação, será aberta a Asset Store dentro do Unity 3D e, então, será possível o download clicando nesta opção. Feito o download, será possível importar as partes desejadas, como mostra a Figura B.2.

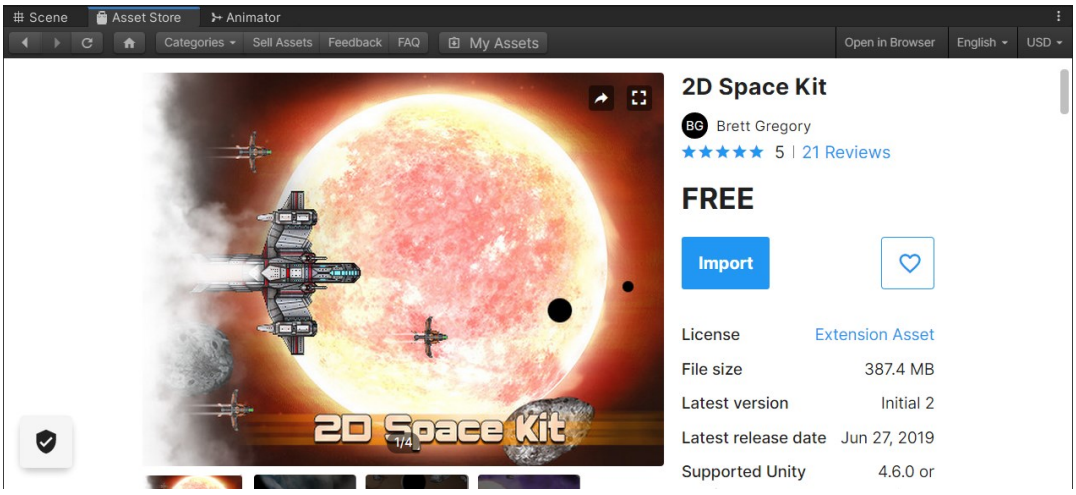

Figura B.2 - Visualização de parte da tela de aquisição do *asset* "*2D Space Kit*" no *Unity 3D*.

Para este *asset*, só serão necessários os elementos marcados nas Figuras B.3 e B.4.

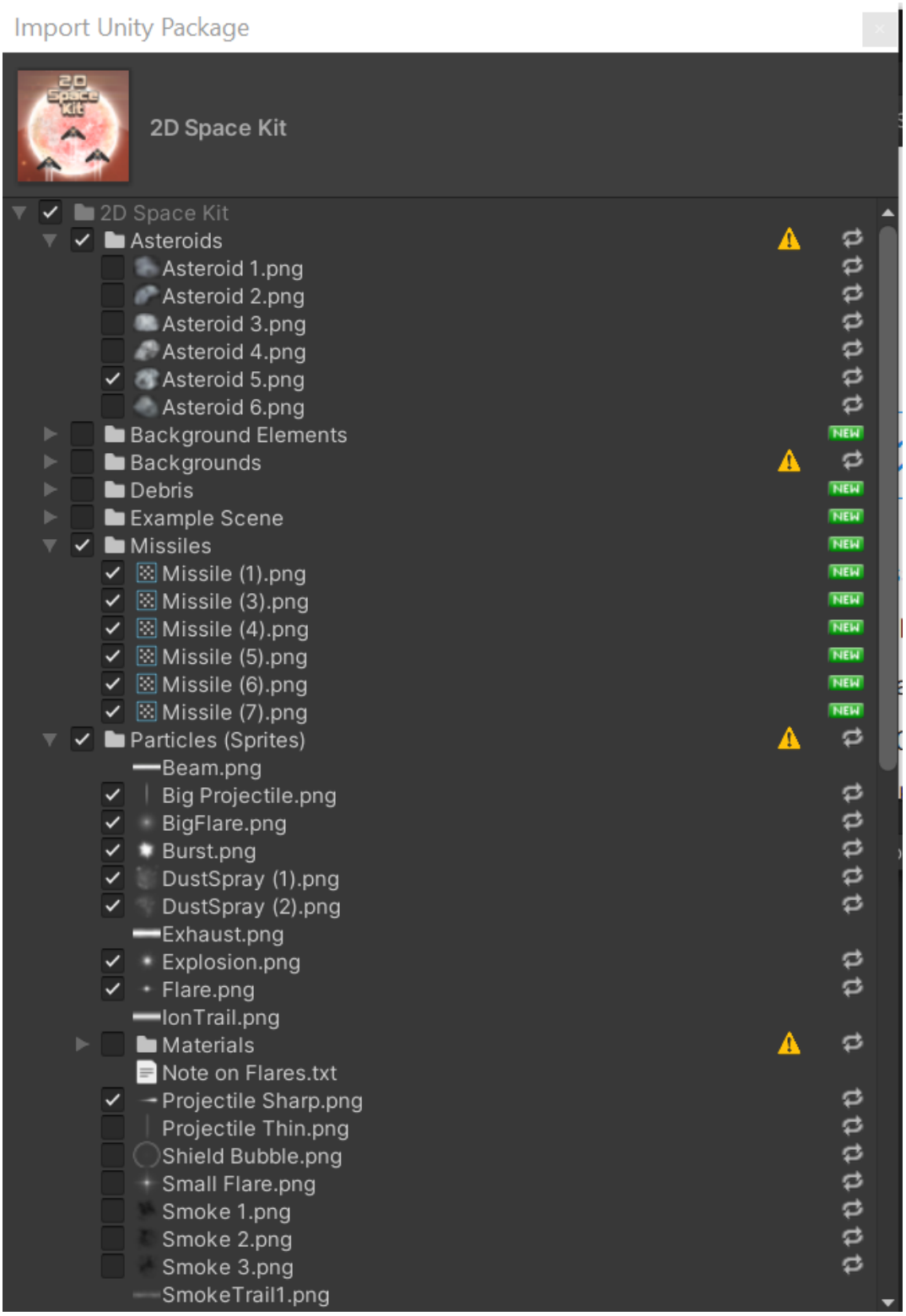

Figura B.3 – Arquivos necessários para o desenvolvimento do jogo.

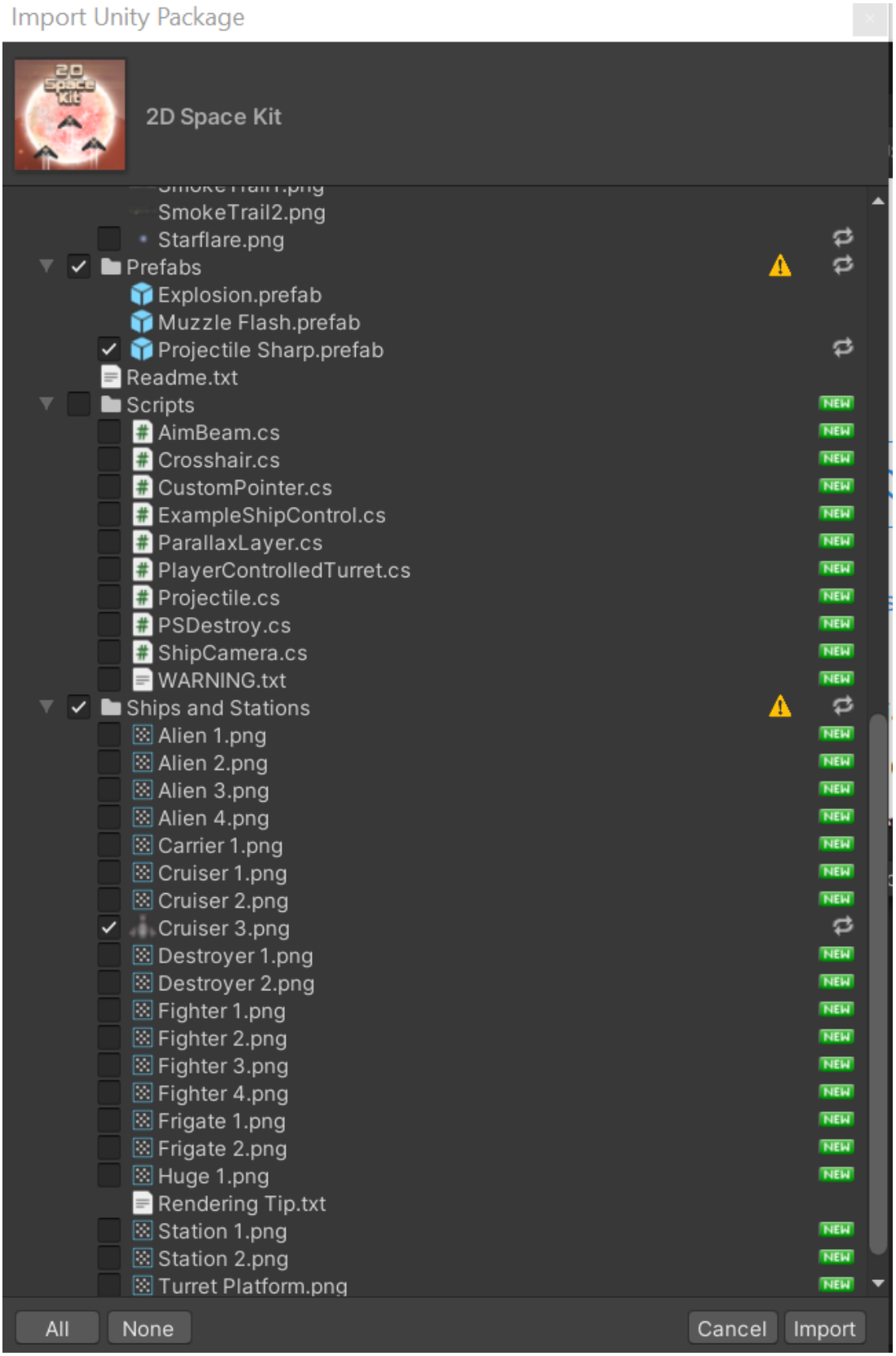

Figura B.4 - Arquivos necessários para o desenvolvimento do jogo, continuação.

C – TUTORIAL PARA CONVERTER UM GIF EM SPRITE

Para este tutorial vamos utilizar o gif que deu origem aos sprites contidos na animação “explodion_0”, utilizada na criação do jogo “Asteroides” e citada no passo 84 de desenvolvimento do mesmo.

Digite no seu *Browser* de internet o nome do *gif*, ou tema que deseja encontrar, e escolha a opção desejada. Para o exemplo em questão, acesse o link: <https://www.kissclipart.com/top-down-explosion-sprite-clipart-sprite-clip-art-c1wfi3/> e baixe gratuitamente o *gif* .

Com o arquivo baixado, abra-o em um conversor GIF → SPRITE, no caso em questão, foi utilizado o conversor online “ezgif” (Disponível em: <https://ezgif.com/gif-to-sprite> - Acesso em 13/10/2021). A tela inicial do site é apresentada na Figura C.1.

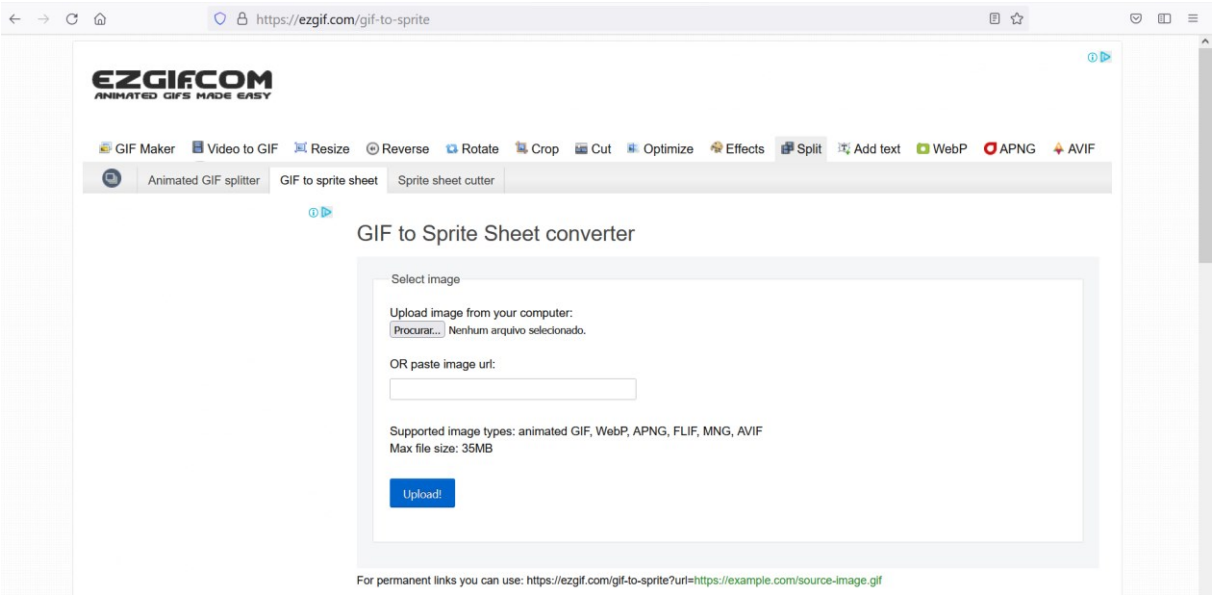

Figura C.1 – Tela inicial do site utilizado para a conversão GIF → SPRITE.

Em “procurar” selecione o *gif* baixado anteriormente e clique em “Abrir”, a Figura C.2 mostra como ficará a tela de seleção de imagem dos site.

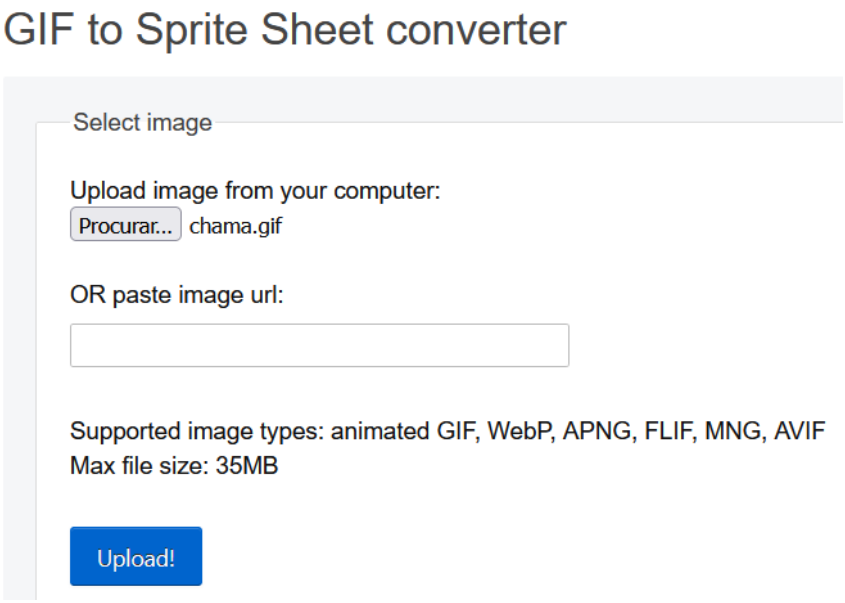

Figura C.2 – Tela de seleção de imagem do site após selecionado o *gif* para conversão.

Após selecionado o *gif*, clique no botão "*Upload*", mostrado na Figura C.2. Configure a página conforme mostra a Figura C.3 e depois clique em "*Convert to Sprite Sheet!*".

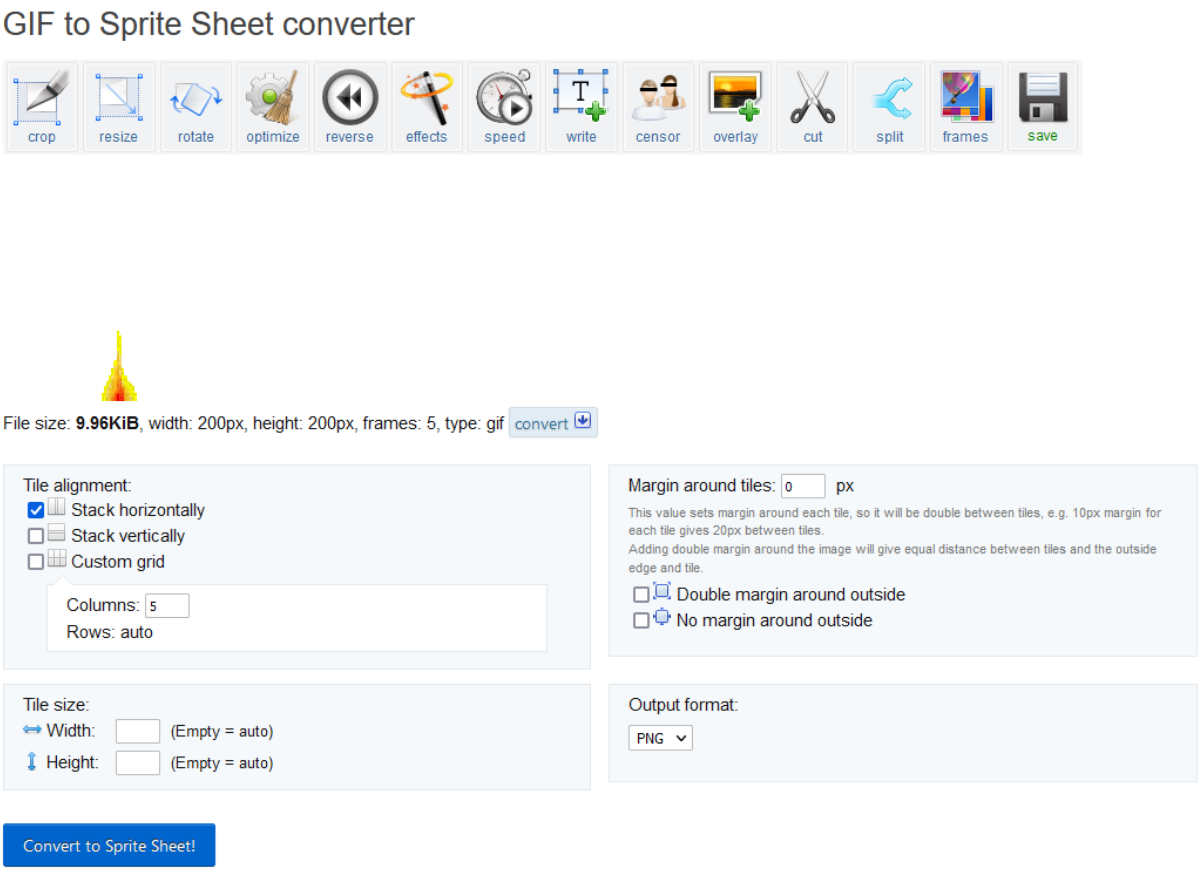

Figura C.3 - Configurando a página de conversão.

Após a conversão, salve em uma pasta o *sprite* obtido clicando no botão “Save”, destacado na Figura C.4, para usar posteriormente na criação do jogo.

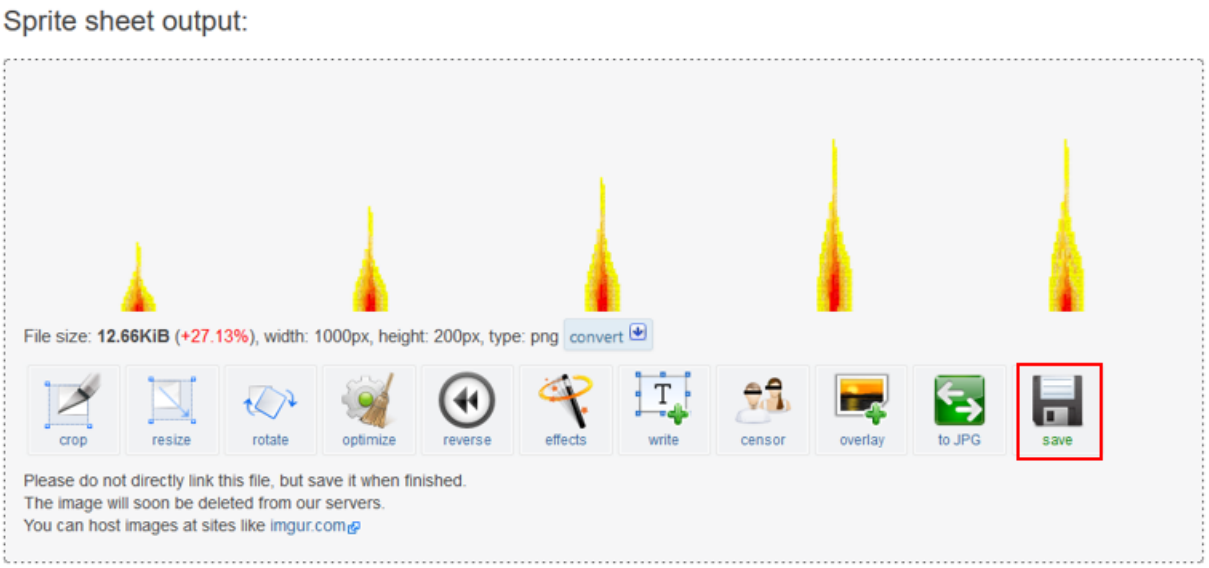

Figura C.4 – Salvando o *sprite* obtido com a conversão.

D – SCRIPT

LevelLoader.cs

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

public class LevelLoader : MonoBehaviour
{
    public GameObject loadingScreen;
    public Slider slider;
    public Text progressText;

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

    IEnumerator LoadAsynchronously (int sceneIndex)
    {
        AsyncOperation operation = SceneManager.LoadSceneAsync(sceneIndex);

        loadingScreen.SetActive(true);

        while (!operation.isDone)
        {
            float progress = Mathf.Clamp01(operation.progress / .9f);

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

            yield return null;
        }
    }
}
```

E - 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 idade;
    public Text genero;
    public Text terapianum;
    public Text inicio;
    public Text hoje;
    public Text caminhoArquivo;

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

        StreamWriter mySW = new StreamWriter(nomeArquivo, true);

        // tópicos básicos do arquivo
        mySW.WriteLine("Nome: " + nome.text);
        mySW.WriteLine("Idade: " + idade.text);
        mySW.WriteLine("Gênero: " + genero.text);
        mySW.WriteLine("Terapia de número: " + terapianum.text);
        mySW.WriteLine("Data de hoje: " + hoje.text);
        mySW.WriteLine("Data de início do tratamento: " + inicio.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);
        }
    }
}
```

F - SCRIPT

SalvarTempoPontuacao.cs

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

public class SalvarTempoPontuacao : MonoBehaviour
{
    string fileName;
    public Text Pontos;
    public Text TempoExecucao;
    public Text Jogo;

    void SaveTP()
    {
        fileName = PlayerPrefs.GetString("fileName");
        StreamWriter mySW = new StreamWriter(fileName, true);
        mySW.WriteLine("Tempo de duração da sessão atual: " + TempoExecucao.text);
        mySW.WriteLine("Pontuação da sessão atual: " + Pontos.text);
        mySW.WriteLine("Jogo: " + Jogo.text);
        mySW.Flush();
        mySW.Close();
    }

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

G -SCRIPT

CenarioInfinito.cs

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

public class CenarioInfinito : MonoBehaviour
{
    public float velocidadeScroll;

    // Update is called once per frame
    void Update()
    {
        Vector2 offset = new Vector2(Time.time * velocidadeScroll, 0);
        GetComponent<Renderer>().material.mainTextureOffset = offset;
    }
}
```

H - SCRIPT

Cruiser.cs

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

public class Cruiser : MonoBehaviour
{
    private Rigidbody2D rb;

    //Use these to get the GameObject's positions
    [System.NonSerialized]
    Vector2 vetor_posicao_pescoco;
    Vector2 vetor_posicao_quadril;

    //You must assign to these two GameObjects in the Inspector
    GameObject pescoco;
    GameObject quadril;
    public GameObject camerinha;

    float AstELD_junta_pescoco_x;
    float AstELD_junta_pescoco_y;
    float AstELD_junta_quadril_x;
    float AstELD_junta_quadril_y;

    void Start()
    {
        rb = GetComponent<Rigidbody2D>();

        //Initialise the Vector
        vetor_posicao_pescoco = Vector2.zero;
        InicializarPreferencias();
    }

    void InicializarPreferencias()
    {
        if (PlayerPrefs.HasKey("ELD_junta_pescoco_x"))
        {
            AstELD_junta_pescoco_x = PlayerPrefs.GetFloat("ELD_junta_pescoco_x");
            //Debug.Log("LE_junta_pescoco_x: " + junta_pescoco_x);
        }
        if (PlayerPrefs.HasKey("ELD_junta_pescoco_y"))
        {
            AstELD_junta_pescoco_y = PlayerPrefs.GetFloat("ELD_junta_pescoco_y");
            Debug.Log("AstELD_junta_pescoco_y: " + AstELD_junta_pescoco_y);
        }
        if (PlayerPrefs.HasKey("ELD_junta_quadril_x"))
        {
            AstELD_junta_quadril_x = PlayerPrefs.GetFloat("ELD_junta_quadril_x");
            //Debug.Log("junta_quadril_x: " + junta_quadril_x);
        }
        if (PlayerPrefs.HasKey("ELD_junta_quadril_y"))
        {
            AstELD_junta_quadril_y = PlayerPrefs.GetFloat("ELD_junta_quadril_y");
            Debug.Log("AstELD_junta_quadril_y: " + AstELD_junta_quadril_y);
        }
```

```
        Debug.Log("SOMA: " + (AstELD_junta_pescoco_y + AstELD_junta_quadril_y/2));
    }

    void Update()
    {
        pescoco = GameObject.FindGameObjectWithTag("pescoco");
        quadril = GameObject.FindGameObjectWithTag("quadril");
        //Fetch the first GameObject's position

        if (quadril != null && pescoco != null)
        {
            vetor_posicao_pescoco = pescoco.transform.position;
            //Fetch the second GameObject's position
            vetor_posicao_quadril = quadril.transform.position;

            //update the position

            if (pescoco.transform.position.y > ((AstELD_junta_pescoco_y +
AstELD_junta_quadril_y) / 4))
            {
                transform.position = new Vector3(-143f, 18.6f,
camerinha.transform.position.z + 499);
                Debug.Log("AstELD_junta_pescocodepoisif_y: " +
AstELD_junta_pescoco_y);
                Debug.Log("AstELD_junta_quadrildepoisif_y: " +
AstELD_junta_quadril_y);
            }
            if (pescoco.transform.position.y < ((AstELD_junta_pescoco_y +
AstELD_junta_quadril_y) / 4) && pescoco.transform.position.y >
((((AstELD_junta_pescoco_y + AstELD_junta_quadril_y) / 2) + AstELD_junta_quadril_y)
/2))
            {
                transform.position = new Vector3(-143f, -23,
camerinha.transform.position.z + 499);
                Debug.Log("AstELD_MEIOOOOOO: " + ((AstELD_junta_pescoco_y +
AstELD_junta_quadril_y)/2));
                Debug.Log("AstELD_ACIMAMEIOMEIO: " + ((AstELD_junta_pescoco_y +
AstELD_junta_quadril_y) / 4));
                Debug.Log("AstELD_ABAIXOMEIO: " + (((AstELD_junta_pescoco_y +
AstELD_junta_quadril_y) / 2) + AstELD_junta_quadril_y) / 2);
            }
            if (pescoco.transform.position.y < ((((AstELD_junta_pescoco_y +
AstELD_junta_quadril_y) / 2) + AstELD_junta_quadril_y) / 2))
            {
                transform.position = new Vector3(-143f, -68.2f,
camerinha.transform.position.z + 499);
                Debug.Log("AstELD_ULTIMO: " + pescoco.transform.position.y);
            }
        }
    }
}
```

I - SCRIPT

spawnbullet.cs

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

public class spawnbullet : MonoBehaviour
{
    public GameObject spawn;
    public float spawnTime;
    public float spawnDelay;

    // Start is called before the first frame update
    void Start()
    {
        InvokeRepeating("Spawn", spawnTime, spawnDelay);
    }

    public void Spawn()
    {
        GameObject clone = (GameObject)Instantiate(spawn, transform.position,
transform.rotation);
        Destroy(clone, 2f);
    }
}
```

J - SCRIPT

Bullet.cs

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

public class Bullet : MonoBehaviour
{
    public float speed = 20f;
    public Rigidbody2D rb;

    // Start is called before the first frame update
    void Start()
    {
        rb.velocity = transform.right * speed;
    }
}
```

K – SCRIPT

spawnAsteroid.cs

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

public class spawnAsteroid : MonoBehaviour
{
    public GameObject spawnee;

    public float spawnTime;
    public float spawnDelay;
    public float spawnWait;
    private int count;

    // Start is called before the first frame update
    void Start()
    {
        InvokeRepeating("SpawnObject", spawnTime, spawnDelay);
    }

    public void SpawnObject()
    {
        Instantiate(spawnee, transform.position, transform.rotation);
    }
}
```

L - SCRIPT

ast.cs

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

public class ast : MonoBehaviour
{
    public GameObject explosao;
    private static int count = 0;
    public Text Asteroides;

    private void Start()
    {
        Asteroides = GameObject.Find("Contador_asteroides").GetComponent<Text>();
        if (Asteroides)
        {
            SetAsteroides();
        }
    }

private void OnTriggerEnter2D(Collider2D other)
    {
        if (other.gameObject.CompareTag("bullet"))
        {
            GameObject e = Instantiate(explosao) as GameObject;
            e.transform.position = transform.position;
            count = count + 1;
            Asteroides.text = "Asteroides: " + count.ToString();
            Destroy(this.gameObject);
            Destroy(other.gameObject);
            Destroy(e.gameObject, 2f);
        }
    }

    void SetAsteroides()
    {
        Asteroides.text = "Asteroides: " + count.ToString();
    }
}
```

M - SCRIPT

Tempo.cs

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

public class Tempo : MonoBehaviour
{
    float tempoSeg = 00;
    int tempoMin = 00, tempoHor = 00;
    string auxiliarSeg, auxiliarMin, auxiliarHor;
    public Text execution;

    // 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 = "Tempo: " + "00:00:" + auxiliarSeg/* + "s"*/;
        }
        else if (tempoHor == 0 && tempoMin != 0)
        {
            execution.text = "Tempo: " + "00:" + auxiliarMin + ":" + auxiliarSeg /*+
"s"*/;
        }
        else
        {
            execution.text = "Tempo: " + auxiliarHor + ":" + auxiliarMin + ":" +
auxiliarSeg/* + "s"*/;
        }
    }
}
```

N - SCRIPT

PlayerCrianca.cs

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

public class PlayerCrianca : MonoBehaviour
{
    private Rigidbody2D rb_menina;

    [System.NonSerialized]
    Vector2 vetor_posicao_pescoco;
    Vector2 vetor_posicao_quadril;
    GameObject pescoco;
    GameObject quadril;
    public GameObject cachorro;
    public GameObject coelho;
    public GameObject gato;
    public int score = 0;
    public Text scoreText;

    public GameObject camerinha;

    float DdPFL_junta_pescoco_x;
    float DdPFL_junta_pescoco_y;
    float DdPFL_junta_quadril_x;
    float DdPFL_junta_quadril_y;

    float DdPEL_junta_pescoco_x;
    float DdPEL_junta_pescoco_y;
    float DdPEL_junta_quadril_x;
    float DdPEL_junta_quadril_y;

    public float speed = 5f;
    public Rigidbody2D rb_cachorro;
    public Rigidbody2D rb_coelho;
    public Rigidbody2D rb_gato;

    private AudioSource audioSource;
    public AudioClip somCachorro;
    public AudioClip somCoelho;
    public AudioClip somGato;

    int ultimaPosicaoMenina = 1;

    void Start()
    {
        rb_menina = GetComponent<Rigidbody2D>();
        vetor_posicao_pescoco = Vector2.zero;
        InicializarPreferencias();
        audioSource = gameObject.GetComponent<AudioSource>();
    }

    void InicializarPreferencias()
    {
        if (PlayerPrefs.HasKey("FL_junta_pescoco_x"))
```

```
        {
            DdPFL_junta_pescoco_x = PlayerPrefs.GetFloat("FL_junta_pescoco_x");
        }
        if (PlayerPrefs.HasKey("FL_junta_pescoco_y"))
        {
            DdPFL_junta_pescoco_y = PlayerPrefs.GetFloat("FL_junta_pescoco_y");
        }
        if (PlayerPrefs.HasKey("FL_junta_quadril_x"))
        {
            DdPFL_junta_quadril_x = PlayerPrefs.GetFloat("FL_junta_quadril_x");
        }
        if (PlayerPrefs.HasKey("FL_junta_quadril_y"))
        {
            DdPFL_junta_quadril_y = PlayerPrefs.GetFloat("FL_junta_quadril_y");
        }

        if (PlayerPrefs.HasKey("EL_junta_pescoco_x"))
        {
            DdPEL_junta_pescoco_x = PlayerPrefs.GetFloat("EL_junta_pescoco_x");
        }
        if (PlayerPrefs.HasKey("EL_junta_pescoco_y"))
        {
            DdPEL_junta_pescoco_y = PlayerPrefs.GetFloat("EL_junta_pescoco_y");
        }
        if (PlayerPrefs.HasKey("EL_junta_quadril_x"))
        {
            DdPEL_junta_quadril_x = PlayerPrefs.GetFloat("EL_junta_quadril_x");
        }
        if (PlayerPrefs.HasKey("EL_junta_quadril_y"))
        {
            DdPEL_junta_quadril_y = PlayerPrefs.GetFloat("EL_junta_quadril_y");
        }
    }

    void Update()
    {
        pescoco = GameObject.FindGameObjectWithTag("pescoco");
        quadril = GameObject.FindGameObjectWithTag("quadril");
        //Fetch the first GameObject's position

        if (quadril != null && pescoco != null)
        {
            vetor_posicao_pescoco = pescoco.transform.position;
            //Fetch the second GameObject's position
            vetor_posicao_quadril = quadril.transform.position;

            //update the position
            if (pescoco.transform.position.x < ((DdPEL_junta_quadril_x +
DdPEL_junta_pescoco_x)/2))
            {
                transform.position = new Vector3(-150.6f, -7f,
camerinha.transform.position.z + 500);
                rb_cachorro.velocity = transform.right * speed;
                rb_coelho.velocity = transform.right * 0;
                rb_gato.velocity = transform.right * 0;

                if (cachorro.transform.position.x >= 113f)
                {
                    cachorro.transform.position = new Vector3(-126.9f,
cachorro.transform.position.y, cachorro.transform.position.z);
                    AtualizaScore();
                }
```

```
                if(ultimaPosicaoMenina != 3)
                {
                    audioSource.Stop();
                    audioSource.clip = somCachorro;
                    audioSource.Play();
                }
                ultimaPosicaoMenina = 3;
            }

            if (pescoco.transform.position.x < ((DdPFL_junta_pescoco_x +
DdPEL_junta_quadril_x) / 2) && pescoco.transform.position.x >
((DdPEL_junta_quadril_x + DdPEL_junta_pescoco_x) / 2))//( DdPEL_junta_quadril_x +
3f) && pescoco.transform.position.x > (DdPEL_junta_quadril_x - 3f))
            {
                transform.position = new Vector3(-150.6f, -31.3f,
camerinha.transform.position.z + 500);
                rb_coelho.velocity = transform.right * speed;
                rb_cachorro.velocity = transform.right * 0;
                rb_gato.velocity = transform.right * 0;

                if (coelho.transform.position.x >= 113f)
                {
                    coelho.transform.position = new Vector3(-126.9f,
coelho.transform.position.y, coelho.transform.position.z);
                    AtualizaScore();
                }

                if (ultimaPosicaoMenina != 2)
                {
                    audioSource.Stop();
                    audioSource.clip = somCoelho;
                    audioSource.Play();
                }
                ultimaPosicaoMenina = 2;
            }

            if (pescoco.transform.position.x > ((DdPFL_junta_pescoco_x +
DdPEL_junta_quadril_x)/2))
            {
                transform.position = new Vector3(-150.6f, -60.8f,
camerinha.transform.position.z + 500);
                rb_gato.velocity = transform.right * speed;
                rb_cachorro.velocity = transform.right * 0;
                rb_coelho.velocity = transform.right * 0;

                if (gato.transform.position.x >= 113f)
                {
                    gato.transform.position = new Vector3(-126.9f,
gato.transform.position.y, gato.transform.position.z);
                    AtualizaScore();
                }

                if (ultimaPosicaoMenina != 1)
                {
                    audioSource.Stop();
                    audioSource.clip = somGato;
                    audioSource.Play();
                }
                ultimaPosicaoMenina = 1;
            }
        }
```

```
        else
        {
            rb_cachorro.velocity = transform.right * 0;
            rb_coelho.velocity = transform.right * 0;
            rb_gato.velocity = transform.right * 0;
            audioSource.Stop();
        }
    }

    public void AtualizaScore()
    {
        score++;
        scoreText.text = score.ToString();
    }
}
```

P - SCRIPT

spawnpescador.cs

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

public class spawnpescador : MonoBehaviour
{
    private Rigidbody2D rb;
    public GameObject player;
    private int count;
    public Text Peixes;
    private float waitTime = 9.0f;
    private float timer = 0.0f;
    private int posicaoAtual;
    private int posicaoAnterior;

    //Use these to get the GameObject's positions
    [System.NonSerialized]
    Vector2 vetorTronco;
    Vector2 vetorPescoco;

    float m_Angle;

    //You must assign to these two GameObjects in the Inspector
    GameObject tronco;
    GameObject pescoco;
    public GameObject camerinha;

    float PescaFQ_junta_pescoco_x;
    float PescaFQ_junta_pescoco_y;
    float PescaFQ_junta_quadril_x;
    float PescaFQ_junta_quadril_y;

    void Start()
    {
        InicializarPreferencias();
        rb = GetComponent<Rigidbody2D>();
        //Initialise the Vector
        vetorTronco = Vector2.zero;
        SetPeixes();
        count = 0;
        posicaoAtual = 1;
        posicaoAnterior = 1;
    }

    void InicializarPreferencias()
    {
        if (PlayerPrefs.HasKey("FQ_junta_pescoco_x"))
        {
            PescaFQ_junta_pescoco_x = PlayerPrefs.GetFloat("FQ_junta_pescoco_x");
        }
        if (PlayerPrefs.HasKey("FQ_junta_pescoco_y"))
        {
            PescaFQ_junta_pescoco_y = PlayerPrefs.GetFloat("FQ_junta_pescoco_y");
```

```
        }
        if (PlayerPrefs.HasKey("FQ_junta_quadril_x"))
        {
            PescaFQ_junta_quadril_x = PlayerPrefs.GetFloat("FQ_junta_quadril_x");
        }
        if (PlayerPrefs.HasKey("FQ_junta_quadril_y"))
        {
            PescaFQ_junta_quadril_y = PlayerPrefs.GetFloat("FQ_junta_quadril_y");
        }
    }

    void Update()
    {
        tronco = GameObject.FindGameObjectWithTag("punho_direito");
        pescoco = GameObject.FindGameObjectWithTag("punho_esquerdo");
        //Fetch the first GameObject's position

        if (tronco != null && pescoco != null)
        {
            vetorTronco = tronco.transform.position;
            //Fetch the second GameObject's position
            vetorPescoco = pescoco.transform.position;

            //update the position
            if (pescoco.transform.position.y > ((PescaFQ_junta_pescoco_y +
PescaFQ_junta_quadril_y) / 4))
            {
                posicaoAtual = 3;

                timer += Time.deltaTime;

                if (timer > waitTime)
                {
                    count = count + 1;
                    timer = 0;
                    SetPeixes();
                    criarPayerPosicaoAtual();
                }
                verificaPosicaoAlterada();
            }

            if (pescoco.transform.position.y < ((PescaFQ_junta_pescoco_y +
PescaFQ_junta_quadril_y) / 4) && pescoco.transform.position.y >
((((PescaFQ_junta_pescoco_y + PescaFQ_junta_quadril_y) / 2) +
PescaFQ_junta_quadril_y) / 2))
            {
                posicaoAtual = 2;

                timer += Time.deltaTime;

                if (timer > waitTime)
                {
                    count = count + 1;
                    timer = 0;
                    SetPeixes();
                    criarPayerPosicaoAtual();
                }
                verificaPosicaoAlterada();
            }

            if (pescoco.transform.position.y < ((((PescaFQ_junta_pescoco_y +
PescaFQ_junta_quadril_y) / 2) + PescaFQ_junta_quadril_y) / 2))
```

```
            {
                posicaoAtual = 1;

                timer += Time.deltaTime;

                if (timer > waitTime)
                {
                    count = count + 1;
                    timer = 0;
                    SetPeixes();
                    criarPayerPosicaoAtual();
                }
                verificaPosicaoAlterada();
            }
        }
    }

    void verificaPosicaoAlterada()
    {
        if (posicaoAtual != posicaoAnterior)
        {
            timer = 0;
            criarPayerPosicaoAtual();
            posicaoAnterior = posicaoAtual;
        }
    }

    void SetPeixes()
    {
        Peixes.text = "Peixes: " + count.ToString();
    }

    void criarPayerPosicaoAtual()
    {
        GameObject newPlayer = (GameObject)Instantiate(player);
        DestroyImmediate(player, true);

        switch (posicaoAtual)
        {
            case 1:
                newPlayer.transform.position = new Vector3(49.3f, -57,
camerinha.transform.position.z + 483);
                break;
            case 2:
                newPlayer.transform.position = new Vector3(49.3f, -7.3f,
camerinha.transform.position.z + 483);
                break;
            case 3:
                newPlayer.transform.position = new Vector3(49.3f, 30.5f,
camerinha.transform.position.z + 483);
                break;
        }

        if (newPlayer != null)
            newPlayer.SetActive(true);

        player = newPlayer;
    }
}
```

Q - SCRIPT

arraia.cs

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

public class arraia : MonoBehaviour
{
    private Rigidbody2D rb;

    //Use these to get the GameObject's positions
    [System.NonSerialized]
    Vector2 vetorTronco;
    Vector2 vetorPescoco;
    float m_Angle;

    //You must assign to these two GameObjects in the Inspector
    GameObject tronco;
    GameObject pescoco;
    public GameObject camerinha;

    void Start()
    {
        rb = GetComponent<Rigidbody2D>();
        //Initialise the Vector
        vetorTronco = Vector2.zero;
    }

    void Update()
    {
        tronco = GameObject.FindGameObjectWithTag("punho_direito");
        pescoco = GameObject.FindGameObjectWithTag("punho_esquerdo");

        if (tronco != null && pescoco != null)
        {
            //Fetch the first GameObject's position
            vetorTronco = tronco.transform.position;
            //Fetch the second GameObject's position
            vetorPescoco = pescoco.transform.position;

            float verticalInput = 40 * Mathf.Atan(pescoco.transform.position.y -
tronco.transform.position.y);

            //update the position
            if (pescoco.transform.position.y < -60)
                transform.position = new Vector3(82.6f, -88.2f,
camerinha.transform.position.z + 483);
            if (pescoco.transform.position.y > -60 && pescoco.transform.position.y <
-20)
                transform.position = new Vector3(82.6f, -38,
camerinha.transform.position.z + 483);
            if (pescoco.transform.position.y > -20)
                transform.position = new Vector3(82.6f, -1.4f,
camerinha.transform.position.z + 483);
        }
    }
}
```

R - SCRIPT

peixe.cs

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

public class peixe : MonoBehaviour
{
    public GameObject spawnee;
    public bool stopSpawing = false;
    public float spawnTime;
    public float spawnDelay;

    // Start is called before the first frame update
    void Start()
    {
        InvokeRepeating("SpawnObject", spawnTime, spawnDelay);
    }

    void SpawnObject()
    {
        GameObject clone = (GameObject) Instantiate(spawnee, transform.position,
Quaternion.identity);
        Destroy(clone, 3.0f);
        if (stopSpawing)
            CancelInvoke("SpawnObject");
    }
}
```

S - SCRIPT

Player.cs

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

public class Player : MonoBehaviour
{
    private CharacterController controller;
    public float score;
    public int scoreCoin;
    public Text scoreText;
    public Text scoreCoinText;
    public float speed;
    public float rayRadius;
    public LayerMask coinLayer;
    public float gravity;

    //Use these to get the GameObject's positions
    [System.NonSerialized]

    Vector2 vetor_posicao_pescoco;
    Vector2 vetor_posicao_quadril;

    GameObject pescoco;
    GameObject quadril;

    public GameObject camerinha;

    private AudioSource audioSource;
    public AudioClip somMoeda;

    float CorLE_junta_pescoco_x;
    float CorLE_junta_pescoco_y;
    float CorLE_junta_quadril_x;
    float CorLE_junta_quadril_y;

    float CorLD_junta_pescoco_x;
    float CorLD_junta_pescoco_y;
    float CorLD_junta_quadril_x;
    float CorLD_junta_quadril_y;

    Vector3 pos = Vector3.zero;
    public float velAnimacao = 1;

    void Start()
    {
        controller = GetComponent<CharacterController>();
        vetor_posicao_pescoco = Vector2.zero;
        audioSource = gameObject.GetComponent<AudioSource>();
        InicializarPreferencias();
    }

    void InicializarPreferencias()
    {
        if (PlayerPrefs.HasKey("LE_junta_pescoco_x"))
```

```
        {
            CorLE_junta_pescoco_x = PlayerPrefs.GetFloat("LE_junta_pescoco_x");
        }
        if (PlayerPrefs.HasKey("LE_junta_pescoco_y"))
        {
            CorLE_junta_pescoco_y = PlayerPrefs.GetFloat("LE_junta_pescoco_y");
        }
        if (PlayerPrefs.HasKey("LE_junta_quadril_x"))
        {
            CorLE_junta_quadril_x = PlayerPrefs.GetFloat("LE_junta_quadril_x");
        }
        if (PlayerPrefs.HasKey("LE_junta_quadril_y"))
        {
            CorLE_junta_quadril_y = PlayerPrefs.GetFloat("LE_junta_quadril_y");
        }

        if (PlayerPrefs.HasKey("LD_junta_pescoco_x"))
        {
            CorLD_junta_pescoco_x = PlayerPrefs.GetFloat("LD_junta_pescoco_x");
        }
        if (PlayerPrefs.HasKey("LD_junta_pescoco_y"))
        {
            CorLD_junta_pescoco_y = PlayerPrefs.GetFloat("LD_junta_pescoco_y");
        }
        if (PlayerPrefs.HasKey("LD_junta_quadril_x"))
        {
            CorLD_junta_quadril_x = PlayerPrefs.GetFloat("LD_junta_quadril_x");
        }
        if (PlayerPrefs.HasKey("LD_junta_quadril_y"))
        {
            CorLD_junta_quadril_y = PlayerPrefs.GetFloat("LD_junta_quadril_y");
        }
    }

    void OnCollision()
    {
        RaycastHit coinHit;

        if (Physics.Raycast(transform.position,
transform.TransformDirection(Vector3.forward + new Vector3(0, 1f, 0)), out coinHit,
rayRadius, coinLayer))
        {
            AddCoin();
            coinHit.transform.gameObject.SetActive(false);
        }
    }

    void Update()
    {
        Vector3 direction = -Vector3.forward * speed;
        OnCollision();
        controller.Move(direction * Time.deltaTime);
        score += Time.deltaTime * 3f;
        scoreText.text = Mathf.Round(score).ToString() + "m";
        pescoco = GameObject.FindGameObjectWithTag("pescoco");
        quadril = GameObject.FindGameObjectWithTag("quadril");
        //Fetch the first GameObject's position

        if (quadril != null && pescoco != null)
        {
            vetor_posicao_pescoco = pescoco.transform.position;
            //Fetch the second GameObject's position
```

```
            vetor_posicao_quadril = quadril.transform.position;

            //update the position
            if (pescoco.transform.position.x < ((CorLE_junta_quadril_x +
CorLE_junta_pescoco_x) / 2))
            {
                pos = new Vector3(-1.3f, camerinha.transform.position.y - 3,
camerinha.transform.position.z - 5);
            }

            if (pescoco.transform.position.x < ((CorLD_junta_pescoco_x +
CorLE_junta_quadril_x) / 2) && pescoco.transform.position.x >
((CorLE_junta_quadril_x + CorLE_junta_pescoco_x) / 2))//( DdPEL_junta_quadril_x +
3f) && pescoco.transform.position.x > (DdPEL_junta_quadril_x - 3f))
            {
                pos = new Vector3(0f, camerinha.transform.position.y - 3,
camerinha.transform.position.z - 5);
            }

            if (pescoco.transform.position.x > ((CorLD_junta_pescoco_x +
CorLE_junta_quadril_x) / 2))
            {
                pos = new Vector3(1.3f, camerinha.transform.position.y - 3,
camerinha.transform.position.z - 5);
            }

            transform.position = Vector3.Lerp(transform.position, pos,
Time.deltaTime * velAnimacao);
        }
    }

    public void AddCoin()
    {
        scoreCoin++;
        scoreCoinText.text = scoreCoin.ToString();
        audioSource.Stop();
        audioSource.clip = somMoeda;
        audioSource.Play();
    }
}
```

T - SCRIPT

SpawnPlatform.cs

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

public class SpawnPlatform : MonoBehaviour
{
    public List<GameObject> platforms = new List<GameObject>();
    public List<Transform> currentPlatforms = new List<Transform>();
    public List<GameObject> coins = new List<GameObject>();
    public float offset;
    private Transform Player;
    private Transform currentPlatformPoint;
    private int platformIndex;
    public GameObject spawnee;
    public int[] platformPositions = {0, 1, 1, 0, 2, 2, 0, 1, 1, 0};

    void Start()
    {
        Player = GameObject.FindGameObjectWithTag("Player").transform;

        for (int i = 0; i < platforms.Count; i++)
        {
            Platform currentPlatfomr = platforms[i].GetComponent<Platform>();
            if (currentPlatfomr)
            {
                currentPlatfomr.RenderIndex(platformPositions[i]);
            }

            Transform p = Instantiate(platforms[i], new Vector3(0, 0, i * (-15.2f)),
transform.rotation).transform;
            currentPlatforms.Add(p);
            offset += 15.2f;
        }
        currentPlatformPoint =
currentPlatforms[platformIndex].GetComponent<Platform>().point;
    }

    // Update is called once per frame
    void Update()
    {
        float distance = Player.position.z - currentPlatformPoint.position.z;
        if (distance <= -5)
        {
            Recycle(currentPlatforms[platformIndex].gameObject, platformIndex);
            platformIndex++;

        if (platformIndex > currentPlatforms.Count - 1)
            {
                platformIndex = 0;
            }

            currentPlatformPoint =
currentPlatforms[platformIndex].GetComponent<Platform>().point;
        }
    }
```

```
    public void Recycle(GameObject platform, int platformIndex)
    {
        platform.transform.position = new Vector3(0, 0, -offset);
        offset += 15.2f;

        Platform currentPlatfomr = platform.GetComponent<Platform>();
        if (currentPlatfomr)
        {
            currentPlatfomr.resetCoins(platformPositions[platformIndex]);
        }
    }
}
```

U - SCRIPT

Platform.cs

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

public class Platform : MonoBehaviour
{
    private Vector3[] offsets = {Vector3.zero,new Vector3(1.3f, 0,0), new Vector3(-
1.3f, 0, 0)};
    public Transform point;
    public GameObject[] coins;

    public void resetCoins(int index)
    {
        for (int i = 0; i < coins.Length; i++)
        {
            coins[i].SetActive(true);
            coins[i].transform.position = new Vector3(offsets[index].x,
coins[i].transform.position.y, coins[i].transform.position.z);
        }
    }

    public void RenderIndex(int index)
    {
        for (int i = 0; i < coins.Length; i++)
        {
            coins[i].transform.position = new Vector3(offsets[index].x,
coins[i].transform.position.y, coins[i].transform.position.z);
        }
    }
}
```

V - SCRIPT

botaocalib.cs

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

public class botaocalib : MonoBehaviour
{
    [System.Serializable]

    public struct PessoaMap
    {
        public string nome;
        public RectTransform transform;
    }

    [SerializeField] PessoaMap[] map;

    float junta_pescoco_x;
    float junta_pescoco_y;
    float junta_quadril_x;
    float junta_quadril_y;

    private void calibrar()
    {
        for (int i = 0; i < map.Length; i++)
        {
            GameObject junta = GameObject.Find(map[i].nome);
            if (junta != null)
            {
                RectTransform transform = junta.GetComponent<RectTransform>();
                if (transform != null)
                {
                    map[i].transform = transform;
                }
            }
        }

        junta_pescoco_x = map[0].transform.position.x;
        junta_pescoco_y = map[0].transform.position.y;
        junta_quadril_x = map[1].transform.position.x;
        junta_quadril_y = map[1].transform.position.y;
    }

    public void botaocalibracaoLateralEsquerda()
    {
        calibrar();
        PlayerPrefs.SetFloat("LE_junta_pescoco_x", junta_pescoco_x);
        PlayerPrefs.SetFloat("LE_junta_pescoco_y", junta_pescoco_y);
        PlayerPrefs.SetFloat("LE_junta_quadril_x", junta_quadril_x);
        PlayerPrefs.SetFloat("LE_junta_quadril_y", junta_quadril_y);
    }

    public void botaocalibracaoLateralDireita()
    {
        calibrar();
        PlayerPrefs.SetFloat("LD_junta_pescoco_x", junta_pescoco_x);
```

```
        PlayerPrefs.SetFloat("LD_junta_pescoco_y", junta_pescoco_y);
        PlayerPrefs.SetFloat("LD_junta_quadril_x", junta_quadril_x);
        PlayerPrefs.SetFloat("LD_junta_quadril_y", junta_quadril_y);
    }

    public void botaocalibracaoFlexaoLombar()
    {
        calibrar();
        PlayerPrefs.SetFloat("FL_junta_pescoco_x", junta_pescoco_x);
        PlayerPrefs.SetFloat("FL_junta_pescoco_y", junta_pescoco_y);
        PlayerPrefs.SetFloat("FL_junta_quadril_x", junta_quadril_x);
        PlayerPrefs.SetFloat("FL_junta_quadril_y", junta_quadril_y);
    }

    public void botaocalibracaoExtensaoLombar()
    {
        calibrar();
        PlayerPrefs.SetFloat("EL_junta_pescoco_x", junta_pescoco_x);
        PlayerPrefs.SetFloat("EL_junta_pescoco_y", junta_pescoco_y);
        PlayerPrefs.SetFloat("EL_junta_quadril_x", junta_quadril_x);
        PlayerPrefs.SetFloat("EL_junta_quadril_y", junta_quadril_y);
    }

    public void botaocalibracaoFlexaoQuadril()
    {
        calibrar();
        PlayerPrefs.SetFloat("FQ_junta_pescoco_x", junta_pescoco_x);
        PlayerPrefs.SetFloat("FQ_junta_pescoco_y", junta_pescoco_y);
        PlayerPrefs.SetFloat("FQ_junta_quadril_x", junta_quadril_x);
        PlayerPrefs.SetFloat("FQ_junta_quadril_y", junta_quadril_y);
    }

    public void botaocalibracaoExtensaoLombarDeitado()
    {
        calibrar();
        PlayerPrefs.SetFloat("ELD_junta_pescoco_x", junta_pescoco_x);
        PlayerPrefs.SetFloat("ELD_junta_pescoco_y", junta_pescoco_y);
        PlayerPrefs.SetFloat("ELD_junta_quadril_x", junta_quadril_x);
        PlayerPrefs.SetFloat("ELD_junta_quadril_y", junta_quadril_y);
    }

    public void Resetar()
    {
        PlayerPrefs.DeleteAll();
    }
}
```

W - SCRIPT

PauseMenu.cs

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

public class PauseMenu : MonoBehaviour
{
    public static bool GameIsPaused = false;
    public GameObject pauseMenuUI;
    // Update is called once per frame
    void Update()
    {
        if (Input.GetKeyDown(KeyCode.Escape))
        {
            if (GameIsPaused)
            {
                Resume();
            }
            else
            {
                Pause();
            }
        }
    }

    public void Resume()
    {
        pauseMenuUI.SetActive(false);
        resetTime();
    }

    void Pause()
    {
        pauseMenuUI.SetActive(true);
        Time.timeScale = 0f;
        GameIsPaused = true;
    }

    public void LoadMenu()
    {
        SceneManager.LoadScene("Menu");
    }

    public void SairMenu()
    {
        Application.Quit();
    }

    public void resetTime()
    {
        Time.timeScale = 1f;
        GameIsPaused = false;
    }
}
```