UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA E
DEPARTAMENTO ACADÊMICO DE ELETRÔNICA
ENGENHARIA DE COMPUTAÇÃO

LUCAS DESTEFANI FABRI

PROTEIN FOLD VR - FERRAMENTA COMPUTACIONAL PARA VISUALIZAÇÃO E MANIPULAÇÃO DE PROTEÍNAS UTILIZANDO UM MODELO COARSE-GRAINED E REALIDADE VIRTUAL EM AMBIENTE GAMIFICADO

TRABALHO DE CONCLUSÃO DE CURSO

CURITIBA

2021

LUCAS DESTEFANI FABRI

PROTEIN FOLD VR - FERRAMENTA COMPUTACIONAL PARA VISUALIZAÇÃO E MANIPULAÇÃO DE PROTEÍNAS UTILIZANDO UM MODELO COARSE-GRAINED E REALIDADE VIRTUAL EM AMBIENTE GAMIFICADO

Trabalho de Conclusão de Curso apresentado ao Departamento Acadêmico de Informática e Departamento Acadêmico de Eletrônica da Universidade Tecnológica Federal do Paraná como requisito parcial para obtenção do grau de Bacharel em Engenharia de Computação.

Orientador: Prof. Dr. César Manuel Vargas Benítez

CURITIBA

2021

LUCAS DESTEFANI FABRI

PROTEIN FOLD VR - FERRAMENTA COMPUTACIONAL PARA VISUALIZAÇÃO E MANIPULAÇÃO DE PROTEÍNAS UTILIZANDO UM MODELO COARSE-GRAINED E REALIDADE VIRTUAL EM AMBIENTE GAMIFICADO

Trabalho de Conclusão de Curso de Graduação apresentado como requisito para obtenção do título de Bacharel em Engenharia de Computação da Universidade Tecnológica Federal do Paraná (UTFPR).

Data de aprovação: 25 de junho de 2021

_____

CÉSAR MANUEL VARGAS BENÍTEZ
Doutorado em Engenharia Elétrica e Informática Industrial pela Universidade Tecnológica Federal do Paraná
Universidade Tecnológica Federal do Paraná (UTFPR)

_____

GUSTAVO BENVENUTTI BORBA
DDoutorado em Engenharia Elétrica e Informática Industrial pela Universidade Tecnológica Federal do Paraná
Universidade Tecnológica Federal do Paraná (UTFPR)

_____

SERGIO MORIBE
Mestrado Profissional em Programa de Pós-Graduação em Engenharia Biomédica pela Universidade Tecnológica Federal do Paraná
Universidade Tecnológica Federal do Paraná (UTFPR)

CURITIBA

2021

AGRADECIMENTOS

Agradeço aos professores Dr. César Manuel Vargas Benítez e Dr. Heitor Silvério Lopes pela cooperação junto ao Laboratório de Bioinformática e Inteligência Computacional (LABIC) da UTFPR - Campus Curitiba. Em especial, ao professor César Benítez, que além de orientar o projeto, teve sua tese de doutorado como guia na fundamentação teórica sobre o dobramento de proteínas e o modelo computacional 3D-AB *off-lattice*. Também agradeço ao Dr. Leandro Takeshi Hattori, na época doutorando, pela sua disposição para esclarecer dúvidas e discutir sobre o dobramento de proteínas e as simulações computacionais, durante meu período de estágio no LABIC. Por fim, agradeço aos meus familiares, que forneceram apoio incondicional durante todo o período do curso, tornando o evento da graduação uma realidade concreta.

RESUMO

DESTEFANI FABRI, Lucas. **Protein Fold VR - Ferramenta Computacional para Visualização e Manipulação de Proteínas utilizando um Modelo Coarse-Grained e Realidade Virtual em Ambiente Gamificado**. 86 f. Trabalho de Conclusão de Curso – Departamento Acadêmico de Informática e Departamento Acadêmico de Eletrônica, Universidade Tecnológica Federal do Paraná. Curitiba, 2021.

As proteínas são estruturas orgânicas fundamentais para a existência da vida. O dobramento de proteínas consiste no processo físico-químico em que uma cadeia de aminoácidos adquire seu formato final, conhecido como estrutura nativa. Uma vez que a função de uma proteína está diretamente relacionada com sua estrutura tridimensional, é na conformação nativa que a proteína exerce a sua função biológica. Na busca por métodos que permitam estudar e simular como este processo ocorre, está compreendido o Problema do Dobramento de Proteínas (*Protein Folding Problem* - PFP), uma das áreas de estudo da biologia computacional mais exploradas atualmente. Sua solução visa permitir uma maior compreensão sobre doenças como Alzheimer e Parkinson, além de auxiliar no desenvolvimento de novos medicamentos. Com isso, o presente relatório descreve o desenvolvimento e discute os resultados obtidos com a ferramenta computacional, pela qual é possível visualizar e manipular a estrutura de uma proteína utilizando um modelo *coarse-grained*, através de uma interface de realidade virtual num ambiente gamificado. A finalidade desta ferramenta é permitir que sejam feitos ajustes pontuais em estruturas previamente processadas por algoritmos de simulação de dobramento de proteínas (disponibilizadas pelo LABIC - Laboratório de Bioinformática e Inteligência Computacional - da UTFPR Curitiba), objetivando encontrar um melhor resultado do ponto de vista da energia interna da estrutura. No desenvolvimento do sistema, foi utilizada a plataforma de desenvolvimento de jogos Unity - inclusive em Realidade Virtual (RV). Também foram incorporados princípios de gamificação, como a interatividade do usuário com o problema e a obtenção de uma recompensa (pontuação). O resultado obtido é um aplicativo para smartphones com o sistema operacional Android e compatíveis com a aplicação Google Cardboard, sendo também necessários óculos de RV (para acomodar o smartphone) e um controle *bluetooth*. Assim, espera-se que a aplicação desperte o interesse pelo estudo do PFP e também auxilie no ensino do processo de dobramento de proteínas com modelos *coarse-grained*.

Palavras-chave: Biologia Computacional. Protein Folding Problem. Realidade Virtual. Gamificação.

ABSTRACT

DESTEFANI FABRI, Lucas. **Protein Fold VR - Computational Tool for Protein Visualization and Manipulation using a Coarse-Grained Model and Virtual Reality in a Gamified Environment**. 86 f. Trabalho de Conclusão de Curso – Departamento Acadêmico de Informática e Departamento Acadêmico de Eletrônica, Universidade Tecnológica Federal do Paraná. Curitiba, 2021.

Proteins are fundamental organic structures to the existence of life. Protein folding is the physicochemical process in which a chain of amino acids takes on its final shape, known as native structure. Since the function of a protein is directly related to its three-dimensional structure, it's in the native conformation that the protein exerts its biological function. In the search for methods to study and simulate how this process occurs, the Protein Folding Problem (PFP) is understood, one of the most explored areas of computational biology today. Its solution aims to allow a greater understanding of disorders such as Alzheimer's and Parkinson's disease, in addition to assisting in the development of new drugs. With this, the present report describes the development and discusses the results obtained with the computational tool, through which it is possible to visualize and manipulate the structure of a protein using a coarse-grained model, through a virtual reality interface in a gamified environment. The purpose of this tool is to allow specific adjustments to be made in structures previously processed by protein folding simulation algorithms (made available by LABIC - Laboratory of Bioinformatics and Computational Intelligence - from UTFPR Curitiba), aiming to obtain a better result, from the point of view of internal energy of the structure. In the system creation, the Unity game development platform was used - also in Virtual Reality (VR). Gamification principles have also been incorporated, such as user interactivity with the problem and achievement of a reward (score). The result obtained is an application for smartphones with the Android operating system and compatible with the Google Cardboard application, requiring VR glasses (to accommodate the smartphone) and a bluetooth control. Thus, it is expected that the application will arouse interest for the study of PFP and also assist in teaching the protein folding process, with coarse-grained models.

Keywords: Computational Biology. Protein Folding Problem. Virtual Reality. Gamification.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE QUADROS

LISTA DE SIGLAS

PSP	*Protein Structure Prediction*
PFP	*Protein Folding Problem*
RV	Realidade Virual
SO	Sistema Operacional
HIV	*Human Immunodeficiency Virus*
CPU	*Central Process Unit*
DNA	*Deoxyribonucleic Acid*
RNA	*Ribonucleic Acid*
RNAm	RNA mensageiro
RNAt	RNA transportador
DM	Dinâmica Molecular
PDB	*Protein Data Bank*
RA	Realidade Aumentada
RM	Realidade Misturada
IHC	Interação Humano-Computador
HMD	*Head-Mounted Display*
6DOF	*Six degrees of freedom*
FPS	*Frames per second*
LABIC	Laboratório de Bioinformática e Inteligência Computacional
SDK	*Software Development Kit*
IDE	*Integrated Development Environment*
MSVC	*Microsoft Visual Studio Community*
API	*Application Programming Interface*
JSON	*JavaScript Object Notation*
PC	*Personal Computer*
USB	*Universal Serial Bus*

LISTA DE SÍMBOLOS

C_α	Carbono alfa
H_2N	Grupo amina
H	Hidrogênio
$COOH$	Grupo carboxila
R	Radical
B	Hidrofílico ou polar
A	Hidrofóbico
C#	C Sharp

SUMÁRIO

1 INTRODUÇÃO

1.1 MOTIVAÇÃO

As proteínas são moléculas compostas por uma cadeia de aminoácidos. Produzidas pelos organismos vivos, atuam nas mais variadas funções biológicas: transporte de oxigênio, ação enzimática e resposta imunológica, sendo fundamentais para o desenvolvimento e manutenção da vida. Somando-se a isso, a função das proteínas está estritamente associada à sua forma, o que torna o estudo do processo de dobramento das cadeias de aminoácidos em sua estrutura nativa (forma de menor energia livre e de caráter atuante dentro dos processos somáticos) uma das áreas mais importantes da biologia computacional (DILL et al., 2008). Isto porque, esse conhecimento é valioso no estudo de doenças como Alzheimer, Parkinson, diabetes mellitus tipo 2, fibrose cística e outras proteinopatias, além de auxiliar no desenvolvimento de novos medicamentos (DILL; MACCALLUM, 2012).

O tema conhecido como dobramento de proteínas, um dos principais problemas da Ciência e que já é estudado há mais de 50 anos, consiste de dois problemas distintos: o *Protein Structure Prediction* (PSP), que busca prever qual a estrutura final de uma dada cadeia polipeptídica. E o *Protein Folding Problem* (PFP), que estuda os comportamentos assumidos pela cadeia de aminoácidos até atingir sua conformação nativa (BENÍTEZ, 2015). A presente proposta busca explorar o PFP por meio da utilização de realidade virtual (RV) e o conceito de gamificação. O uso de uma interface de RV, tecnologia que está se difundindo rapidamente, e dos preceitos da gamificação, tem por objetivo propiciar ao usuário uma experiência mais interativa e lúdica.

Portanto, o que motiva o desenvolvimento do presente projeto é a utilização dos conhecimentos adquiridos durante o curso de formação em Engenharia de Computação no desenvolvimento de uma abordagem didática para um problema relevante da bioinformática.

1.2 RESULTADOS ESPERADOS

A partir da ideia do jogo Foldit[1], será construída uma ferramenta computacional que permita a visualização e a manipulação de estruturas de proteínas que foram previamente processadas por meio de algoritmos de Dinâmica Molecular, de maneira a permitir um ajuste na estrutura, diminuindo sua energia livre e fornecendo uma resposta mais próxima da estrutura ideal. A utilização da interface de RV e dos conceitos de gamificação visam, também, uma aplicabilidade didática, fomentando o interesse dos que tiverem contato com a aplicação pelas áreas da bioinformática e computação.

1.3 OBJETIVOS

1.3.1 OBJETIVO GERAL

Desenvolver um sistema computacional para visualizar e manipular a estrutura tridimensional de proteínas (utilizando um modelo *coarse-grained*) por meio de uma interface de realidade virtual em um ambiente gamificado.

1.3.2 OBJETIVOS ESPECÍFICOS

- Estudar o modelo matemático adotado para a representação das estruturas proteicas e cálculo de energia interna da molécula.
- Determinar uma base de estruturas com as quais o usuário poderá interagir.
- Desenvolver um método para converter os dados da molécula para um formato compatível com a aplicação.
- Desenvolver uma aplicação para representação das estruturas proteicas permitindo a sua visualização e manipulação.
- Determinar métodos para realizar controle de colisão e cálculo de energia interna da estrutura molecular.
- Integrar o software com uma interface de realidade virtual, que consiste em óculos e controle manual.
- Testar a usabilidade e desempenho da aplicação.

[1] https://fold.it/

1.4 ESCOPO

O objetivo do projeto é desenvolver uma ferramenta computacional que permita a visualização e manipulação de estruturas tridimensionais, que representam simplificadamente uma molécula proteica, por meio de uma interface de RV que consistirá em um dispositivo para visualização (*headset*) e um método de controle manual. Para isso, pretende-se utilizar preferencialmente a plataforma Unity[2] e a linguagem C# para a construção de uma aplicação para RV que seja executada preferivelmente em um dispositivo smartphone rodando o sistema operacional (SO) Android e compatível com a plataforma Google Cardboard[3].

Do ponto de vista físico, dá-se preferência a kits de RV que consistem em um dispositivo *bluetooth* para ser utilizado como controle manual e um *headset* para acoplamento de um smartphone. Isso porque, esse tipo de equipamento apresenta um menor custo em relação aos dispositivos *head-mounted display*, tais como Oculus Rift[4] e HTC Vive[5].

1.5 TRABALHOS RELACIONADOS

Uma inspiração, ao construir um aplicativo no formato de jogo que auxiliasse a pesquisa na área do dobramento de proteínas, advém da aplicação Foldit[6]. Esse jogo é disponibilizado para as plataformas Microsoft Windows, macOS e Linux e por meio dos dados coletados foi publicado um artigo referente ao estudo do vírus da imunodeficiência humana (HIV) (KHATIB et al., 2011). Outro projeto com características similares é o Folding@home [7]. Essa aplicação utiliza processamento distribuído em plataforma *desktop* (Microsoft Windows, macOS e Linux) para realizar simulações de dobramento de proteínas, sem a necessidade de intervenção do usuário. A proposta é a de que qualquer pessoa, com uma CPU ociosa, pode auxiliar em novas descobertas científicas para o tratamento de doenças como o câncer.

[2] https://unity.com/
[3] https://arvr.google.com/cardboard/
[4] https://www.oculus.com/rift/
[5] https://www.vive.com/us/
[6] https://fold.it/
[7] https://foldingathome.org/

2 FUNDAMENTAÇÃO TEÓRICA

A fim de tornar mais familiar ao leitor os temas discutidos no projeto e embasar os métodos determinados para cumprir os objetivos pretendidos, será apresentada uma breve fundamentação teórica acerca de: conceitos biológicos de proteínas, PFP, realidade virtual e gamificação.

2.1 PROTEÍNAS

Dentre as moléculas que constituem os organismos vivos, as proteínas são, quimicamente, as que possuem estrutura mais complexa e funcionamento mais sofisticado. Além disso, estes compostos orgânicos constituem a maior parte da massa seca de uma célula (15% em geral) e são responsáveis pelas suas funcionalidades e estrutura (ALBERTS et al., 2017; NELSON; COX, 2014). Os papéis desempenhados pelas proteínas são diversos, muitas atuam como enzimas na catálise de reações metabólicas. Outras, são responsáveis pelo transporte de pequenas moléculas através da membrana celular. Tem-se, também, as transportadoras de nutrientes, como a hemoglobina. E, ainda, há as proteínas especializadas, que atuam como hormônios, anticorpos, toxinas, fibras elásticas, fibras de sustentação, anticongelantes e fonte de bioluminescência (ALBERTS et al., 2017; COOPER; HAUSMAN, 2007).

2.1.1 SÍNTESE PROTEICA

As proteínas são sintetizadas a partir das informações armazenadas na molécula de DNA (*deoxyribonucleic acid* - em português, ácido desoxirribonucleico) presente nas células. Nas células humanas, o processo geral de síntese de uma proteína inicia-se no núcleo da célula, com a leitura dos nucleotídios de um gene, que é um trecho de DNA responsável pela expressão de alguma característica hereditária (GRIFFITHS et al., 2016). A esse processo é dado o nome de transcrição e ao final é formada uma fita de RNA (*ribonucleic acid* - em português, ácido ribonucleico) mensageiro (RNAm), composto por bases nitrogenadas complementares

às presentes no DNA (GRIFFITHS et al., 2016). O Quadro 1 apresenta essas relações de complementaridade entre as bases nucleicas, dentro da dupla fita de DNA e na leitura do DNA pelo RNAm.

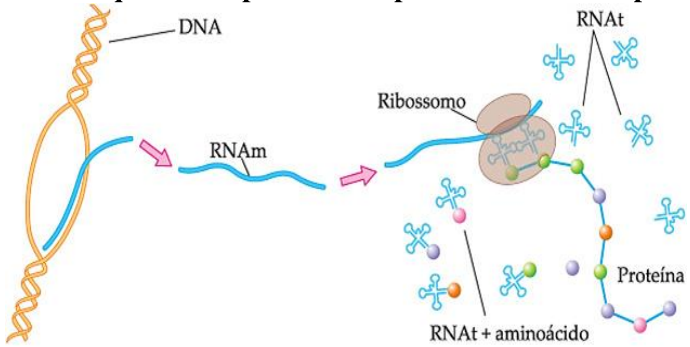
Quadro 1: Relações de complementaridade entre as bases nucleicas.

Base Nitrogenada		Base Complementar		
Nome	Código	DNA - DNA	DNA - RNAm	RNAm - RNAt
Adenina	A	A - T	A - U	A - U
Timina	T	T - A	T - A	-
Uracila	U	-	-	U - A
Citosina	C	C - G	C - G	C - G
Guanina	G	G - C	G - C	G - C

Fonte: Adaptado de (LODISH et al., 2016).

Na sequência, o RNAm é levado para os ribossomos, organelas celulares especializadas na síntese proteica, onde a fita de RNAm é então utilizada como referência para o processo de tradução. A tradução consiste na formação de uma cadeia de aminoácidos tendo como molde os códons (trincas de bases nitrogenadas) presentes no RNAm. Desse modo, cada códon é pareado com o seu anticódon situado numa molécula de RNA transportador (RNAt) (ver relação no Quadro 1) ligada à um dos vinte aminoácidos mais comuns (GRIFFITHS et al., 2016). Deste processo resulta um polímero linear, que após sofrer a ação de forças não covalentes, provenientes da interação com o meio e entre os próprios elementos da cadeia, num processo conhecido como dobramento, alcança uma estrutura final que apresenta atividade bioquímica, recebendo então o nome de proteína (LODISH et al., 2016). A Figura 1 apresenta um panorama geral do processo de síntese proteica.

Figura 1: Esquema simplificado do processo de síntese proteica.

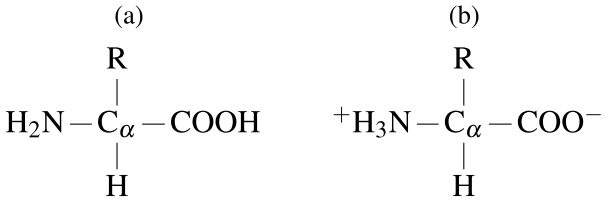


Fonte: Adaptado de https://slideplayer.com.br/slide/15844837/.

2.1.2 AMINOÁCIDOS

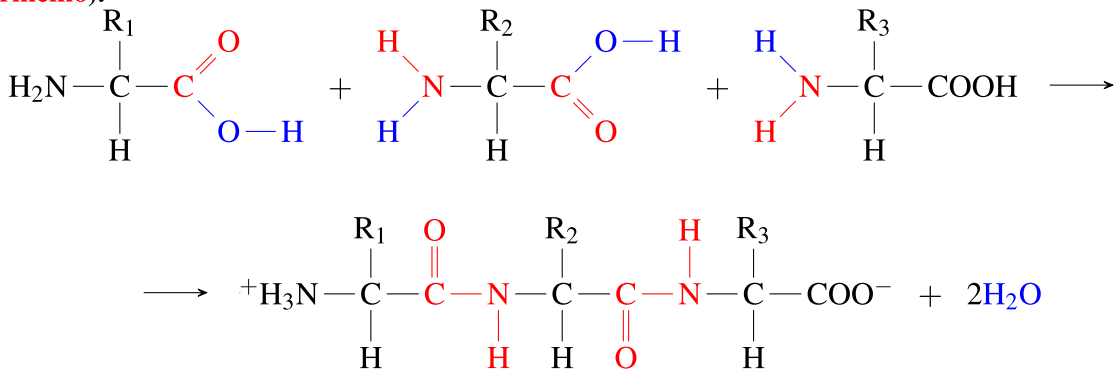
Os aminoácidos da cadeia polimérica proteica são compostos orgânicos formados por um carbono central, denominado de carbono alfa (C_α), que realiza quatro ligações covalentes: com um grupo amina (H_2N), com um átomo de hidrogênio (H), com um grupo carboxila ($COOH$) e com um radical (R) (GRIFFITHS et al., 2016), como pode ser visualizado na Figura 2. As ligações peptídicas originadas pela reação de síntese, por desidratação do grupo amina de um aminoácido com o grupamento carboxila de outro (Figura 3), constituem a interação direta, ligação química covalente, que mantém um resíduo de aminoácido ligado a outro, conferindo à proteína uma estrutura primária linear (LODISH et al., 2016).

Figura 2: (a) Estrutura geral de um aminoácido. (b) Aminoácido ionizado em meio aquoso e pH 7.



Fonte: (a) Adaptado de (GRIFFITHS et al., 2016). (b) Adaptado de (ALBERTS et al., 2017).

Figura 3: Reação de síntese por desidratação (em azul) formando ligações peptídicas (em vermelho).

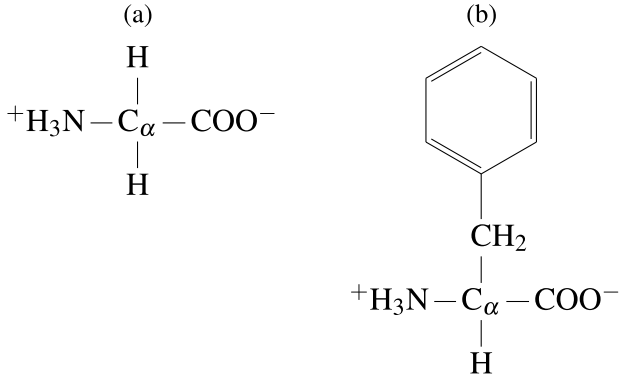


Fonte: Adaptado de (LODISH et al., 2016) e (ALBERTS et al., 2017).

Dos aminoácidos presentes na natureza, vinte são os mais utilizados pelos diferentes seres vivos para produzir proteínas. O que difere um aminoácido do outro é o radical R (COOPER; HAUSMAN, 2007). Os radicais, também chamados de cadeia lateral, são uma ramificação da cadeia principal e apresentam diferentes tipos de estrutura química, como pode ser visto na Figura 4. Com isso, uma importante característica das cadeias laterais dos

aminoácidos é a sua hidrofobicidade. Se suas moléculas sofrem repulsão em meio aquoso, como gotas de óleo que se agrupam quando dispersas em água formando estruturas condensadas, é chamada de hidrofóbica. Caso seja atraída pela molécula de água, diz-se que a molécula é hidrofílica (ALBERTS et al., 2017). Uma característica do grupamento *R* que determina esse comportamento é a polaridade de sua molécula. Moléculas polares, ou com carga, apresentam comportamento hidrofílico (B), como a água, que é uma molécula polar. Moléculas apolares, por sua vez, apresentam comportamento hidrofóbico (A) (NELSON; COX, 2014; STILLINGER; HEAD-GORDON, 1995). O Quadro 2 apresenta os vinte aminoácidos, com suas respectivas hidrofobicidades classificadas segundo (ALBERTS et al., 2017).

Figura 4: Exemplos de aminoácidos: (a) Glicina e (b) Fenilalanina.



Fonte: Adaptado de (NELSON; COX, 2014).

Quadro 2: Classificação dos aminoácidos de acordo com a hidrofobicidade.

Aminoácido	Código	Cadeia lateral	Hidrofobicidade	
Alanina	Ala	Apolar	A	Hidrofóbicos
Glicina	Gly	Apolar	A	
Valina	Val	Apolar	A	
Leucina	Leu	Apolar	A	
Isoleucina	Ile	Apolar	A	
Prolina	Pro	Apolar	A	
Fenilalanina	Phe	Apolar	A	
Metionina	Met	Apolar	A	
Triptofano	Trp	Apolar	A	
Cisteína	Cys	Apolar	A	
Ácido aspártico	Asp	Negativa	B	Hidrofílicos
Ácido glutâmico	Glu	Negativa	B	
Arginina	Arg	Positiva	B	
Lisina	Lys	Positiva	B	
Histidina	His	Positiva	B	
Asparagina	Asn	Polar não carregada	B	
Glutamina	Gln	Polar não carregada	B	
Serina	Ser	Polar não carregada	B	
Treonina	Thr	Polar não carregada	B	
Tirosina	Tyr	Polar não carregada	B	

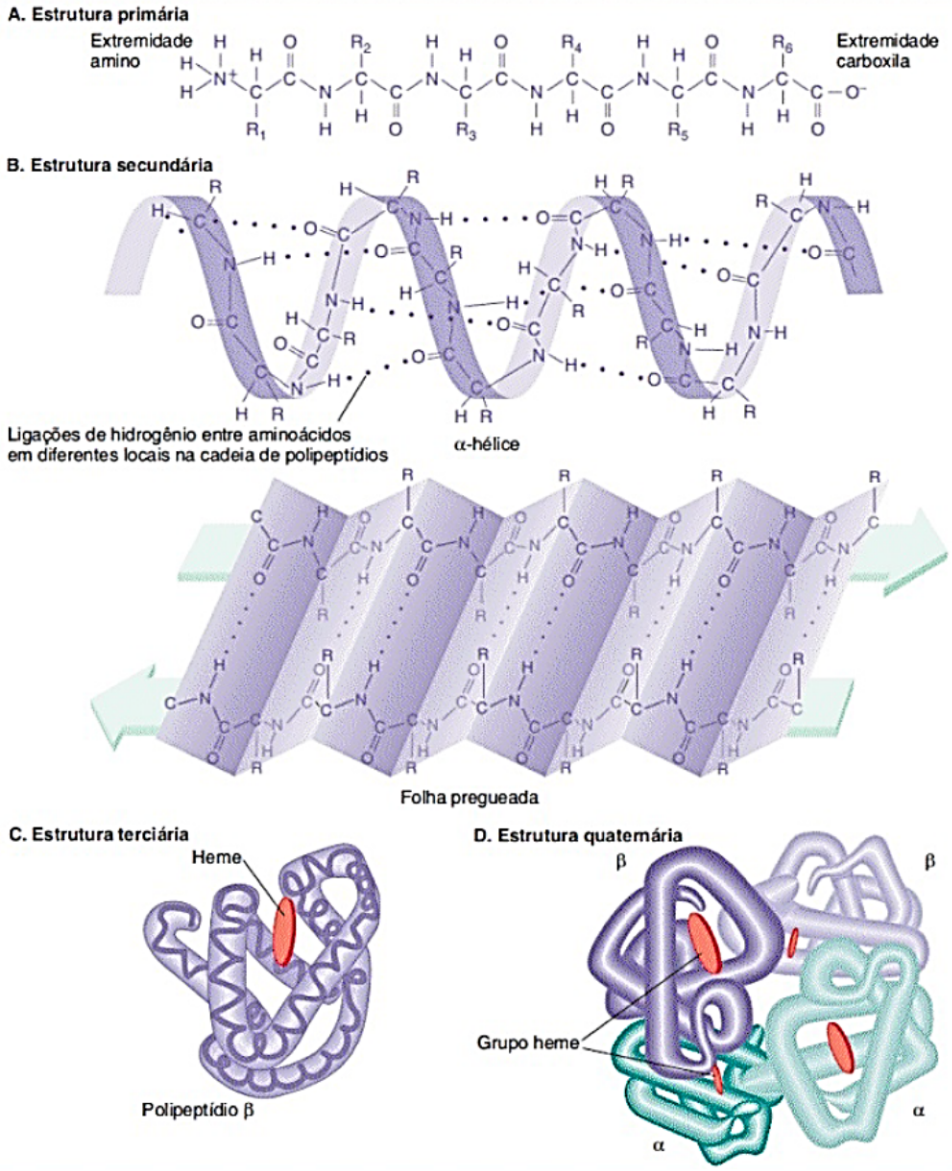
Fonte: Adaptado de (ALBERTS et al., 2017) e (BENÍTEZ, 2015).

2.1.3 DOBRAMENTO DE PROTEÍNAS

A função de uma proteína está diretamente relacionada à sua forma. E há fortes indícios de que a forma de uma proteína é predeterminada pela sequência de aminoácidos que a compõe (ANFINSEN, 1973). Isto porque, logo que ocorre a formação de um polipeptídeo no meio intracelular, formado em média por 70% de água (NELSON; COX, 2014), forças não covalentes começam a agir sobre ele num processo conhecido como dobramento ou enovelamento de proteínas (em inglês, *protein folding*). Essas forças podem ser classificadas em quatro tipos: ligações de hidrogênio, atrações eletrostáticas, atrações de Van der Waals e combinação de forças hidrofóbicas. Apesar dessas forças serem mais fracas que as ligações covalentes (de 30 à 300 vezes), quando atuando em conjunto, elas podem unir fortemente regiões da cadeia, mesmo entre polipeptídeos distintos, contribuindo assim para o dobramento das proteínas (ALBERTS et al., 2017).

Após o dobramento, o polipeptídeo atinge uma forma tridimensional estável, conhecida como conformação nativa, com sítios reativos em sua superfície e sendo, assim, chamado de proteína. No entanto, esse é um processo complexo e seu estudo é classificado em quatro estágios, de acordo com a evolução da estrutura ao longo do dobramento do polipeptídeo (NELSON; COX, 2014). A cadeia linear de aminoácidos recebe o nome de estrutura primária. A estrutura secundária é o resultado da evolução do dobramento da estrutura primária, sendo possível observar padrões estruturais, mesmo em sequências diferentes. Os principais padrões conhecidos são as α-hélices, as β-folhas e os *turns*. Na Figura 5, pode ser observada a representação esquemática de uma α-hélice e de uma β-folha (folha pregueada). A estrutura tridimensional estável de uma única cadeia principal caracteriza sua estrutura terciária e nas proteínas formadas pelo agrupamento de mais de um polipeptídeo, o resultado dessa interação entre estruturas tridimensionais ainda confere uma estrutura quaternária à proteína (NELSON; COX, 2014). A Figura 5 ilustra as diferentes estruturas de uma proteína.

Figura 5: Representação esquemática do dobramento de proteínas e das estruturas intermediárias. A estrutura quaternária é referente à uma molécula de hemoglobina, formada por quatro subunidades polipeptídicas.



Fonte: Adaptado de (GRIFFITHS et al., 2016).

Um dos aspectos mais importantes do estudo do dobramento de proteínas é o fato de que muitas doenças estão relacionadas a falhas nesse processo. Nesse âmbito, tem-se como exemplo as doenças: Alzheimer (ALBERTS et al., 2017), Parkinson (ALBERTS et al., 2017), Huntington (RAMASWAMY et al., 2007; IMARISIO et al., 2008), diabetes tipo 2 (AGUZZI; O'CONNOR, 2010), fibrose cística (FRASER-PITT; O'NEIL, 2015) e doenças priônicas como doença de Creutzfeldt-Jakob e encefalopatia espongiforme bovina, mais conhecida como doença da vaca louca (ALBERTS et al., 2017).

2.2 PROBLEMA DO DOBRAMENTO DE PROTEÍNAS

A determinação de como uma sequência de aminoácidos interage com o meio intracelular e assume (ou não) uma estrutura tridimensional estável consiste em parte do PFP (DILL et al., 2008). Nem todas as cadeias de aminoácidos tornam-se proteínas. Considerando os vinte aminoácidos que as constituem, a quantidade de cadeias possíveis cresce exponencialmente na proporção de 20^n (sendo n o tamanho da cadeia). Sabe-se que o tamanho das proteínas, em geral, varia entre 50 e 2000 aminoácidos (ALBERTS et al., 2017). E, exemplificando para um polipeptídeo com trezentos aminoácidos, tem-se 20^{300} possibilidades. No entanto, esse valor é tão grande, que para produzir um exemplar de cada molécula, não existiriam átomos suficientes no universo (ALBERTS et al., 2017). Com isso, infere-se que nem todas as sequências de aminoácidos dão origem a proteínas. Apenas uma fração desse conjunto de possibilidades produz estruturas estáveis e funcionais.

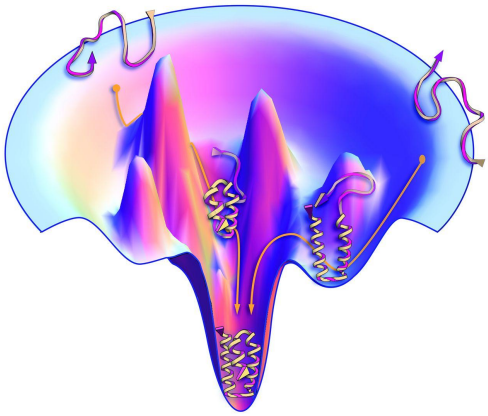
Na década de 1960, um dos estudos que lançou luz à esse processo foi (ANFINSEN et al., 1961). Nele, por meio de reagentes químicos, uma proteína foi desnaturada e observaram-se alterações em sua estrutura tridimensional. Quando reestabelecidas as condições fisiológicas (com a retirada do desnaturante), espontaneamente esta retornou a seu estado nativo. Esse resultado permitiu supor que a informação para o polipeptídeo alcançar a estrutura nativa é preexistente, ou seja, a sequência de aminoácidos é o que determinaria a conformação nativa da proteína (ANFINSEN, 1973). Desse modo, foi formulada por Christian Anfinsen a Hipótese Termodinâmica, sugerindo que a estrutura tridimensional de uma proteína no estado nativo é aquela que apresenta o menor valor de energia livre de Gibbs (ANFINSEN, 1973).

Consonante a essa linha de raciocínio, tem-se os argumentos propostos por Cyrus Levinthal em (LEVINTHAL, 1968). Nesse estudo, foi proposto que é impossível uma proteína testar todas as suas possíveis conformações para encontrar o estado de menor energia. A fim de exemplificar, supondo uma proteína com N aminoácidos que podem assumir μ orientações no espaço, tem-se μ^N conformações possíveis. Considerando apenas duas possibilidades de orientação e uma proteína com 100 resíduos, tem-se $2^{100} \approx 10^{30}$ conformações (CONTESSOTO et al., 2018). Tomando um tempo médio da ordem de um picossegundo (10^{-12}) para encontrar uma conformação, seriam necessários aproximadamente 10^{18} segundos para alcançar todas as 10^{30} conformações (BENÍTEZ, 2015). No entanto, esse tempo é da ordem de grandeza da idade do universo (10^{10} anos).

Portanto, isso caracteriza um paradoxo, conhecido como paradoxo de Levinthal, pois o tempo que uma proteína demora para se dobrar é da ordem de alguns segundos, menor que o tempo para encontrar o estado de menor energia global (LEVINTHAL, 1968; ZWANZIG et al., 1992). Uma consequência desse paradoxo é a possibilidade da existência de um caminho cinético preferencial, a partir do qual uma cadeia polipeptídica primária num estado aleatório, seguindo uma sequência de passos intermediários, atingisse seu estado nativo (LEVINTHAL, 1968; ZWANZIG et al., 1992).

Alternativamente, no final da década de 1980, surgiu uma teoria que propunha a existência de uma superfície de energia, ao invés de um caminho único sucessivo (CONTESSOTO et al., 2018). Segundo essa teoria, o dobramento segue uma paisagem energética afunilada, na qual, nas extremidades superiores, encontram-se as estruturas iniciais aleatórias com maior energia livre. Desse modo, a medida que o dobramento ocorre, observam-se conjuntos de configurações cada vez mais reduzidos, ocupando níveis energéticos mais brandos, até o alcance do estado nativo (LEOPOLD et al., 1992). Essas novas abordagens deram ao estudo do dobramento, das etapas intermediárias e dos mecanismos envolvidos no processo, maior atenção dos pesquisadores. A Figura 6 apresenta um exemplo conceitual de funil de dobramento. O estreitamento representa uma restrição no espaço conformacional de busca, para alcançar o estado nativo, conforme o processo de dobramento evolui.

Figura 6: Modelo de paisagem de dobramento em formato de funil.



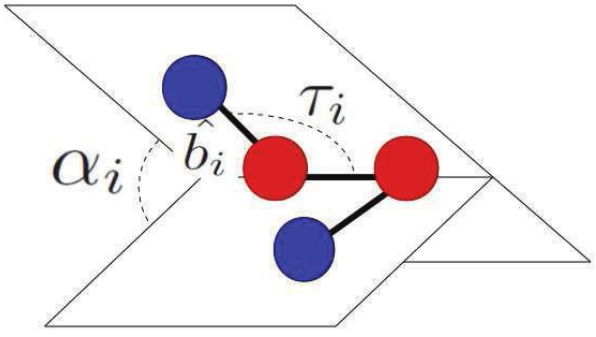
Fonte: (DILL; MACCALLUM, 2012).

2.2.1 MODELOS DE PROTEÍNAS

Uma vez que o dobramento de proteínas é um fenômeno biológico, o estudo do PFP necessita que modelos sejam criados, por meio do conhecimento físico, químico e matemático, na tentativa de explicar, prever e simular o dobramento de proteínas. Os diferentes modelos de dobramento podem ser classificados de acordo com parâmetros como a mobilidade espacial, a posição entre os resíduos e a sua granularidade. Quanto a mobilidade espacial: bidimensionais (2D) ou tridimensionais (3D). Os modelos em rede (*lattice*) admitem apenas ângulos restritos entre as partículas, ao passo que os modelos contínuos (*off-lattice*) propiciam maior liberdade de movimento. A granularidade está relacionada à maneira como os elementos constituintes da cadeia polipeptídica estão representados. Há modelos atômicos, nos quais cada átomo e as forças envolvidas estão descritos da maneira mais fiel possível. Por outro lado, nos modelos *coarse-grained*, simplificações podem sintetizar todo um aminoácido numa única característica, como a hidrofobicidade, por exemplo. Todos esses fatores devem ser considerados, uma vez que têm impacto direto no custo computacional da execução de algoritmos de simulação (NGO et al., 1994).

O modelo 3D-AB *off-lattice*, proposto por Irbäck, é um modelo tridimensional (3D), contínuo (*off-lattice*) e simplificado (*coarse-grained*). Esse modelo reduz os vinte diferentes aminoácidos à partículas pontuais, localizadas na posição do C_α, com massa unitária e que realizam ligações de comprimento unitário, representados por $\widehat{b}_i$, como pode ser visto na Figura 7. Utilizando uma escala de hidrofobicidade como a descrita no Quadro 2, os resíduos admitem os valores: hidrofóbico (A) ou hidrofílico (B), na computação da energia interna da estrutura (IRBÄCK et al., 1997; STILLINGER; HEAD-GORDON, 1995). A expressão matemática descrita na Equação 1 apresenta o valor da energia interna de uma molécula utilizando o modelo 3D-AB *off-lattice* (IRBÄCK et al., 1997).

Figura 7: Elementos do modelo 3D-AB off-lattice.



Fonte: (BENÍTEZ, 2015)

$$E_{Potential} = E_{Angles} + E_{Tortion} + E_{Lennard-Jones} \tag{1}$$

$$E_A = -k_1 \sum_{i=1}^{N-2} \widehat{b}_i \cdot \widehat{b}_{i+1} \tag{2}$$

$$E_T = -k_2 \sum_{i=1}^{N-3} \widehat{b}_i \cdot \widehat{b}_{i+2} \tag{3}$$

$$E_{LJ} = \sum_{i=1}^{N-2} \sum_{j=i+2}^{N} 4\varepsilon(\sigma_i, \sigma_j)(r_{ij}^{-12} - r_{ij}^{-6}) \tag{4}$$

Nas Equações 2 e 3 tem-se $k_1 = -1$ e $k_2 = \frac{1}{2}$ (IRBÄCK et al., 1997). Na Equação 4 o termo r_{ij} representa a distância entre dois resíduos i e j, e o potencial $\varepsilon(\sigma_i, \sigma_j)$ é definido como:

$$\varepsilon(\sigma_i, \sigma_j) = \begin{cases} 1, & \text{se AA} \\ \frac{1}{2}, & \text{se AB, BA ou BB} \end{cases} \tag{5}$$

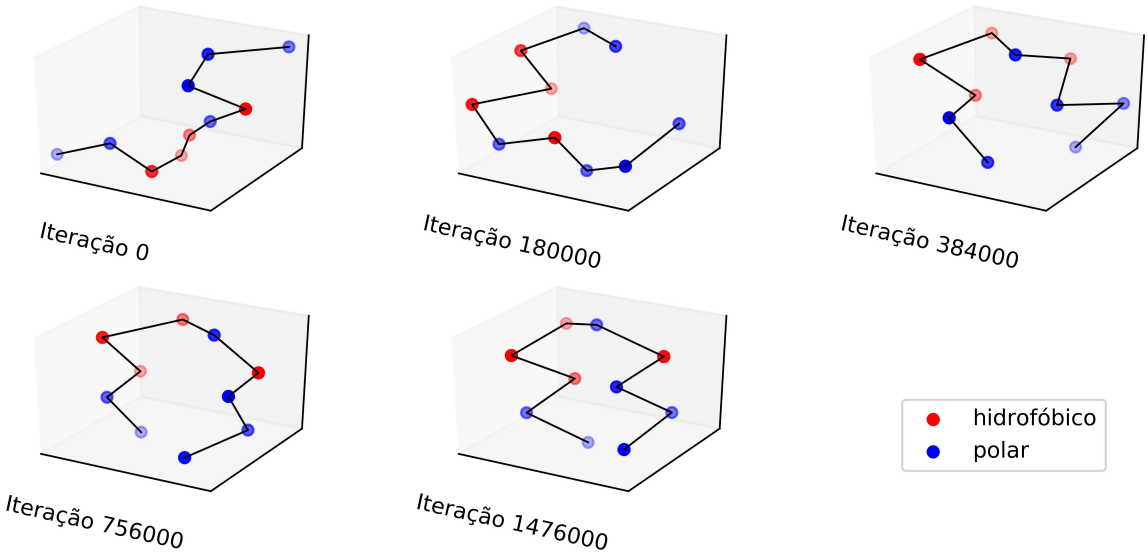
2.2.2 DINÂMICA MOLECULAR

A dinâmica molecular (DM) é um método de simulação computacional que visa recriar a trajetória espacial de corpos por meio da integração numérica das leis clássicas do movimento (BENÍTEZ, 2015). A segunda lei de Newton é a principal expressão computada (RAPAPORT et al., 1996), mas o método também admite expressões específicas para cada fenômeno.

As simulações em DM são determinísticas, isto é, a partir de um ponto inicial, uma partícula sempre irá percorrer a mesma trajetória, diferentemente de métodos estocásticos como o de Monte Carlo (BENÍTEZ, 2015). No entanto, o método de DM é computacionalmente custoso, o que faz com que modelos *coarse-grained* sejam indicados em situações nas quais os recursos de hardware são menos disponíveis, como é o caso das CPUs para *desktops*, por exemplo (NGO et al., 1994). A Figura 8 exemplifica uma sequência de dobramento obtida por meio de simulação computacional utilizando o algoritmo proposto por (BENÍTEZ; LOPES, 2012; BENÍTEZ; LOPES, 2013). Ainda, a Figura 9 apresenta uma comparação entre uma estrutura obtida por meio de dinâmica molecular com modelo *coarse-grained* e uma obtida por métodos com menor granularidade, a partir do *Protein Data Bank*[1] (PDB).

[1] http://www.rcsb.org/

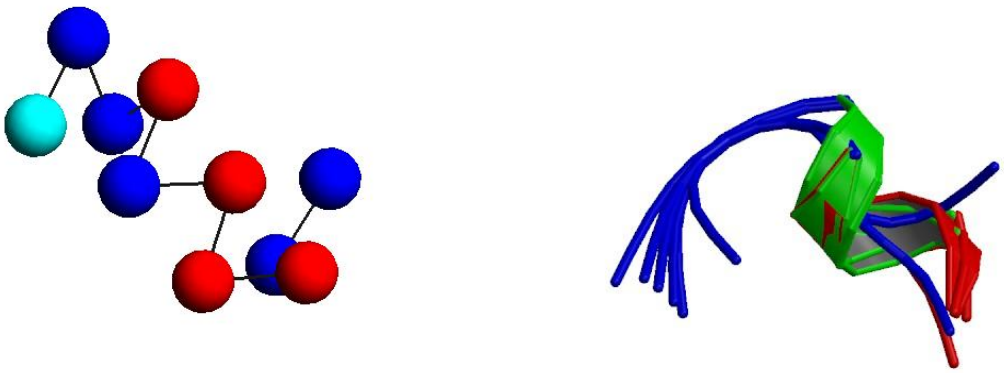
Figura 8: ***Pathway*** **de dobramento da proteína 1KUW, processado com o algoritmo de DM proposto em (BENÍTEZ; LOPES, 2012; BENÍTEZ; LOPES, 2013).**



Fonte: Autoria própria.

Figura 9: Representação do dobramento da proteína 1KUW: (a) por meio do algoritmo de DM de (BENÍTEZ; LOPES, 2012; BENÍTEZ; LOPES, 2013) e (b) apresentada pelo PDB.

(a) (b)



Fonte: (a) Autoria própria. (b) Imagem do RCSB PDB de ID 1KUW (MASCIONI et al., 2003).

2.3 PRINCÍPIOS DE REALIDADE VIRTAL

O termo *"Virtual Reality"* - adotado em português, na sua literalidade, como Realidade Virtual (RV) - foi utilizado primeiramente por Jaron Lanier, no início dos anos 1980, para diferenciar simulações tradicionais feitas por um computador daquelas realizadas por múltiplos usuários em um ambiente compartilhado (MACHOVER; TICE, 1994; ARAUJO; KIRNER, 1996; ZHENG et al., 1998; NETTO et al., 2002). Outro conceito de RV foi o citado por Steve Bryson, como sendo o uso de computadores e interfaces com o usuário para criar mundos tridimensionais, com objetos interativos e com uma forte sensação de presença tridimensional (BRYSON, 1996). E ainda, a RV é comumente utilizada para aumentar a imersão do usuário no ambiente, facilitando a compreensão e possibilitando uma interação mais intuitiva com o cenário e seus objetos, aproximando a experiência de uso para algo mais próximo do mundo real.

Além disso, é comum encontrar outros termos relacionados à RV, como: Realidade Aumentada (RA) e Realidade Misturada (RM), vindos do inglês *Augmented Reality* e *Mixed Reality*, respectivamente. Desse modo, a RV se aplica aos sistemas nos quais a percepção sensorial do usuário é total ou parcialmente suprimida, para então ser substituída por um conteúdo virtual (ZHENG et al., 1998; KIRNER; SISCOUTTO, 2007). Já na RA, o ambiente real no qual o usuário se encontra mantém-se e é enriquecido ou melhorado, por meio de artifícios tecnológicos. A exemplo, tem-se o conceito de óculos semitransparentes que apresentam informações na forma de mapas ou de diagramas, sobre o ambiente físico visualizado pelo usuário (KIRNER; SISCOUTTO, 2007; NETTO et al., 2002). A RM, por sua vez, é uma ideia que permeia a RA e a RV, de modo que a natureza virtual dos elementos introduzidos pela interface é reativa ao ambiente (Microsoft, 2020; MILGRAM; KISHINO, 1994). Por exemplo, ao projetar um vaso de flores (virtual) sobre uma mesa (real), se o usuário tocasse o vaso e o empurrasse até o final da mesa, este seria - virtualmente - derrubado no chão, em contraste com a RA, na qual a mão do usuário transpassaria a projeção do objeto.

Complementarmente, as características imersão, interação e envolvimento são recorrentes na bibliografia relacionada. De modo que, a imersão apresenta-se como a capacidade do sistema em fazer o usuário crer que pertence ao ambiente criado artificialmente (ZHENG et al., 1998; NETTO et al., 2002). A interação, por sua vez, reflete-se na capacidade dos elementos do ambiente virtual reagirem às ações do usuário, captadas pelos dispositivos de entrada (ZHENG et al., 1998; NETTO et al., 2002). E, por fim, o envolvimento resume o interesse do usuário em participar do ambiente virtual, seja de maneira passiva, como espectador, ou ativa, tomando parte nas ações realizadas na RV (NETTO et al., 2002).

Ademais, a fim de proporcionar aos usuários a experiência de um ambiente virtual, são necessário elementos apropriados para viabilizar a Interação Humano-Computador (IHC), tanto para as entradas de usuário quanto para o retorno do sistema (NETTO et al., 2002; KIRNER; SISCOUTTO, 2007). Os dispositivos de entrada são necessários para que o usuário possa interagir com o ambiente virtual. Sendo assim, pode-se citar como dispositivos de entrada os sensores utilizados para obtenção da orientação e trajetória, como acelerômetros e giroscópios acoplados ao dispositivo de visualização, sendo que também podem ser utilizados sistemas com infravermelho, ultrassom ou câmeras externas (NETTO et al., 2002). Também, tem-se objetos como as *datagloves*, luvas especiais que captam os movimentos da mão do usuário e os traduzem em sinais elétricos que podem ser lidos pela aplicação (NETTO et al., 2002). Há, ainda, sensores que captam o movimento das mãos por meio de câmeras e os convertem em uma entrada digital, como é o caso do Leap Motion Controler[2]. Além disso, podem ser utilizados controles manuais do tipo *joystick*, como os que acompanham os dispositivos Oculus Rift[3], HTC Vive[4] e Samsung Gear VR[5]. Por fim, há as esteiras omnidirecionais, que atuam tanto como dispositivo de entrada quanto de retorno. Nelas é possível que o usuário desloque-se em qualquer direção, por uma distância indeterminada e sem correr o risco de colidir com objetos no ambiente físico em que se encontra.

As tecnologias de retorno, ou saída, baseiam-se no isolamento externo e posterior ativação dos sentidos visual e auditivo, principalmente, sendo que o tato também é explorado, geralmente em aplicações mais específicas (NETTO et al., 2002). Do ponto de vista dos dispositivos visuais, o mais comum é o tipo *Head-Mounted Display* (HMD). Esse consiste em um par de visores para projeção binocular estereoscópica, podendo ser um *hardware* dedicado, como é o caso do Oculus Rift e HTC Vive, ou um smartphone montado em um suporte, como o Samsung Gear VR e o Google Cardboard[6]. Todos esses dispositivos estão ilustrados na Figura 10. Quanto ao som, para obter um efeito 3D, deve-se utilizar sistemas de áudio capazes de simular o estímulo de uma fonte sonora localizada no espaço tridimensional, o que é possível com as placas de som utilizadas em computadores atuais (KIRNER; SISCOUTTO, 2007). Por fim, existem ainda os dispositivos hápticos, que atuam na percepção tátil. Esse tipo de retorno procura transmitir ao usuário sensações da resistência mecânica e da textura de um elemento virtual (KIRNER; SISCOUTTO, 2007). O emprego desse tipo de dispositivo é muito pertinente em áreas como a médica, de modo que os profissionais possam treinar procedimentos com maior fidelidade, sem o risco de causar dano (ZHENG et al., 1998; NETTO et al., 2002).

[2]https://www.ultraleap.com/product/leap-motion-controller/
[3]https://www.oculus.com/rift/
[4]https://www.vive.com/us/
[5]https://www.samsung.com/global/galaxy/gear-vr/
[6]https://arvr.google.com/cardboard/

Figura 10: *Headsets* de RV: (a) Oculus Rift S, (b) HTC Vive Pro, (c) Samsung Gear VR e (d) Google Cardboard.

(a) (b)

(c) (d)

Fonte: (a) https://www.oculus.com/rift-s/
(b) https://www.vive.com/us/product/vive-pro-eye/overview/
(c) https://www.samsung.com/africa_pt/wearables/gear-vr-r325/
(d) https://arvr.google.com/cardboard/manufacturers/

Nos dispositivos de interface visual de RV com maior difusão no mercado atual (Oculus Rift, HTC Vive, Samsung Gear VR, Google Cardboard e afins), o mais comum é a utilização de um sistema de projeção de imagens estereoscópico em conjunto com um sistema de detecção dos movimentos da cabeça (*head tracking*). Isso significa que cada olho enxerga uma imagem ligeiramente diferente, que precisa ser renderizada separadamente (PIMENTEL; TEIXEIRA, 1993). Quanto ao rastreamento, o mais comum

é a utilização de um sensor com ao menos seis graus de liberdade (6DOF - *six degrees of freedom*), possibilitando rastrear movimentos translacionais e rotacionais em todas as direções do espaço tridimensional (NETTO et al., 2002). No entanto, um problema muito comum entre os usuário de dispositivos visuais de RV é a cinetose ou *motion sickness*, como também é conhecida. Essa condição é resultado da diferença de estímulos recebidos pelo cérebro, em relação ao que os olhos veem e o sistema vestibular no ouvido interno percebe (responsável pelo equilíbrio e percepção de movimento). Assim, o organismo, primando pela autopreservação, por acreditar que está sendo envenenado, provoca reações de náusea, vertigem e eventualmente vômito (Wareable, 2016). Fatores que ajudam a reduzir o problema são uma elevada taxa de quadros (*frames per second* - FPS), acima de 60 Hz, e a redução da latência entre a imagem e os movimentos realizados pelo usuário. Além disso, pesquisas com a finalidade de reduzir esse efeito estão sendo fomentadas pela indústria de RV, uma vez que esse inconveniente prejudica a difusão dessa tecnologia (Wareable, 2016).

Por fim, as tendências para o futuro da RV são promissoras. Uma vez que possui um amplo espectro de aplicação, pode-se encontrar tecnologias de RV em diversas áreas, como: medicina, entretenimento, ensino e aprendizagem, aviação, aeroespacial, militar, robótica, engenharia, arquitetura e outras (NETTO et al., 2002; KIRNER; SISCOUTTO, 2007). Grandes empresas do setor de tecnologia como IBM, Google, Facebook e Microsoft tem investido cifras na casa dos milhões de dólares no desenvolvimento de tecnologias de RV, RA e RM (Wired, 2016). Ainda, o Facebook está testando a versão beta de um ambiente virtual chamado Horizon[7], no qual os usuários podem participar de uma RV colaborativa. E, também, com a miniaturização dos chips de processamento, dispositivos de HDM que antes eram grandes e pesados a ponto de serem utilizados suspensos com contrapesos, hoje podem ser substituídos pelo próprio smartphone do usuário, capaz de realizar tanto o processamento da aplicação quanto da renderização gráfica. Com isso, percebe-se o promissor potencial de crescimento e difusão das tecnologias de RV para as próximas décadas.

2.4 PRINCÍPIOS DE GAMIFICAÇÃO

O jogo é inerente ao homem. Assim afirmou o historiador holandês Johan Huizinga (1872-1945), para o qual há nos seres vivos, incluso o ser humano, um instinto que impele os indivíduos às atividades lúdicas, incentivando o relacionamento e a preparação para a realização de tarefas complexas, entre outras habilidades (HUIZINGA, 1971). Com isso, e a crescente acessibilidade à internet e aos aparelhos eletrônicos (smartphones, tablets, computadores

[7] https://www.oculus.com/facebook-horizon/

pessoais e consoles de videogames), a difusão na sociedade dos jogos digitais (denominados *games* em inglês) não parece um fato tão surpreendente. Sendo, inclusive, parte da realidade do Brasil, onde segundo estatísticas (Prodview, 2020), em 2020, 84 milhões de habitantes - valor superior a um terço da população - foram usuários de jogos eletrônicos, principalmente nas plataformas *on-line*.

Sendo assim, mesmo havendo estudos desde os anos 1980, a fim de tornar as interfaces de usuário mais amigáveis em videogames (MALONE, 1981), foi somente a partir de 2010 que iniciou-se uma discussão mais voltada à utilização de jogos eletrônicos para fins não apenas recreativos (MCGONIGAL, 2010). E, dessa procura por abordar o assunto mais seriamente, expressa em trabalhos como (MCGONIGAL, 2011), surgiu a necessidade de definir o fenômeno da gamificação (derivado do inglês *gamification*) de produtos e serviços (DETERDING et al., 2011a).

Desse modo, uma proposta acadêmica, amplamente citada para definir o termo, é apresentada por Sebastian Deterding, em (DETERDING et al., 2011b), como: *"Gamification is the use of game design elements in non-game contexts"*. O que numa tradução livre seria: "Gamificação é o uso de elementos de design de jogos em contextos de não jogo". Para não tornar o conceito vago, os próprios autores salientam que existem quatro pontos que precisam ser esclarecidos.

Primeiramente, a ideia de jogo (*game*), está relacionada a uma atividade lúdica, com um sistema de regras claro, em que o esforço competitivo dos envolvidos tem em vista um objetivo, uma meta (SALEN; ZIMMERMAN, 2003). Ainda, quanto a esse conceito de jogo (*game*), é necessário diferenciá-lo da brincadeira (*play*), sendo que essa tem caráter mais livre, descomprometido, diferentemente daquele, mais restrito e objetivo (CAILLOIS; BARASH, 2001).

Quanto aos elementos a que Deterding et al. (2011b) se refere, é feita uma comparação com os jogos sérios (*serious games*), sendo que essas aplicações são, de fato, um jogo completo mas sem o propósito exclusivo do entretenimento. Sendo assim, em contraponto, existem aplicações que incorporam elementos de jogos, são gamificadas (*"gamified"*), sem constituir um jogo propriamente, ao menos do ponto de vista de seus usuários. A partir disso, pode-se destacar algumas características que são elementos intrínsecos dos jogos, como: pontuação, níveis de dificuldade e tabelas de ranqueamento (ZICHERMANN; CUNNINGHAM, 2011). Desse modo, Deterding conclui restringindo esses elementos àqueles que são característicos de um jogo, ou seja, que em conjunto caracterizam uma aplicação como jogo, mas que ao serem implementados individualmente não tornam a aplicação um jogo propriamente dito.

Do ponto de vista do design, a respeito dos elementos incorporados às aplicações gamificadas, existem situações em que tecnologias originadas nas plataformas de jogos são empregadas em outros ambientes, como, por exemplo, a utilização de controles de videogames ou a mecânica 3D de jogo em ferramentas de visualização para fins científicos ou gráficos (DETERDING et al., 2011a). Sendo assim, faz-se necessário diferenciar a gamificação da simples utilização de tecnologias oriundas de jogos eletrônicos. Em sua descrição, Deterding et al. (2011b) também menciona diferentes níveis de abstração dos elementos de design de jogos, sendo eles: padrões de interface de design; padrões de design de jogo ou mecânica de jogo; princípios de design, heurísticas e "lentes"; modelos conceituais de unidades de design de jogos e, por fim, processos e métodos de design de jogos. Sendo os últimos mais abstratos que os primeiros (mais concretos).

Por fim, Deterding esclarece sobre o contexto de não jogo (*"non-game context"*). Sua intenção com esse termo é não restringir o uso da gamificação à áreas exclusivas, fora do escopo do entretenimento e diversão, como somente a educação e o ensino, por exemplo.

Assim, os resultados esperados ao se utilizar elementos de jogos para gamificar uma aplicação são: uma maior motivação e engajamento dos usuários, em virtude do reforço positivo proporcionado por pontos e recompensas, maior resiliência, por não considerar as derrotas como um problema insolúvel, e aperfeiçoamento do trabalho cooperativo (MCGONIGAL, 2011; LIU et al., 2011; GROH, 2012). Como exemplos, pode-se citar aplicações como: Duolinguo[8] - plataforma de ensino de idiomas; Chore Wars[9] - aplicação que ludifica as tarefas domésticas; Foursquare[10] - rede social de compartilhamento da geolocalização e Foldit[11] - jogo que aproxima o PFP à usuários leigos, tendo sido publicado um artigo sobre suas contribuições no estudo de proteínas relacionadas ao HIV (KHATIB et al., 2011).

[8] https://pt.duolingo.com/
[9] http://www.chorewars.com/
[10] https://foursquare.com/
[11] https://fold.it/

3 DESENVOLVIMENTO

Esta seção apresenta a metodologia de desenvolvimento, um diagrama geral da aplicação e descreve os aspectos relacionados à construção da aplicação, sendo esses: a plataforma de desenvolvimento, a interface de realidade virtual, a implementação do modelo de estrutura proteica, a conversão entre o formato proveniente do algoritmo de DM em uma estrutura jogável e as formas de interação do usuário com a aplicação.

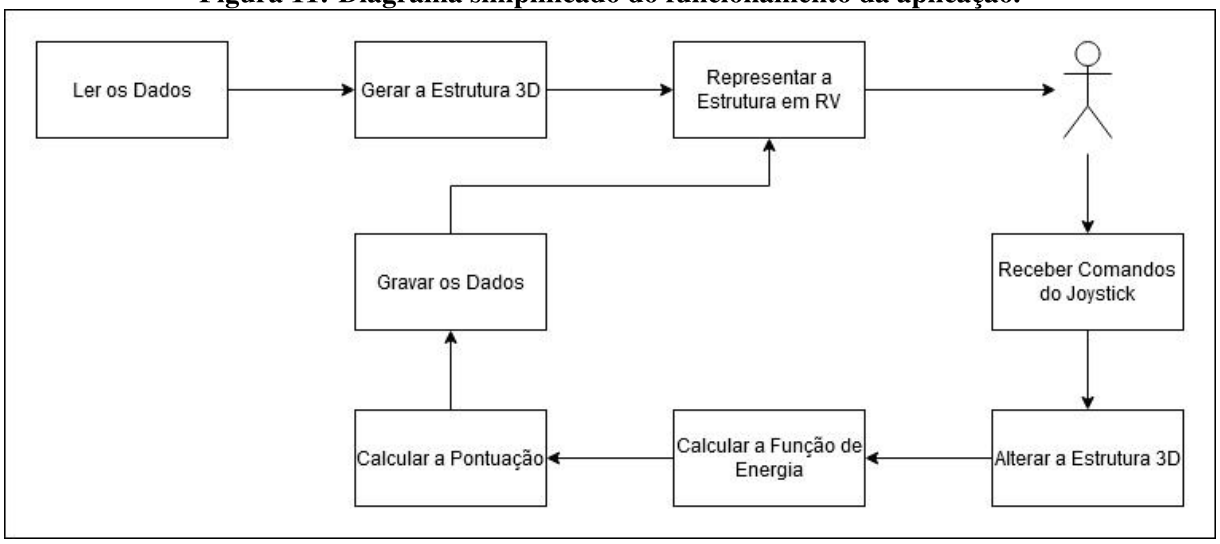
3.1 METODOLOGIA

A metodologia adotada para a realização do projeto seguiu uma sequência de etapas predeterminada. Primeiramente, realizou-se o levantamento da fundamentação teórica acerca dos temas de PFP, RV e gamificação, a fim de conhecer as particularidades do modelo para a representação das moléculas proteicas e qual o ambiente de desenvolvimento preferível para aplicações em RV. Em seguida, após estudo da tecnologia, teve-se como objetivo iniciar a modelagem da aplicação, utilizando a plataforma de desenvolvimento Unity. Na sequência, foram selecionadas as estruturas pré-processadas para configurar o conjunto de entradas padrão da aplicação, oriundas de métodos de DM, obtidas junto ao Laboratório de Bioinformática e Inteligência Computacional (LABIC) da UTFPR Curitiba, em colaboração com pesquisa realizada previamente pelo professor Dr. César Benítez (BENÍTEZ; LOPES, 2012; BENÍTEZ; LOPES, 2013; HATTORI et al., 2017). Em posse dos dados, estes foram importados para a aplicação RV sob a forma de estruturas manipuláveis. Após a sua utilização na aplicação, converteu-se essas estruturas novamente para o seu formato de dados original, possibilitando sua utilização por pessoal da área de bioinformática e demais interessados.

3.2 DIAGRAMA

O diagrama na Figura 11 apresenta a sequência de ações que caracterizam o funcionamento da aplicação.

Figura 11: Diagrama simplificado do funcionamento da aplicação.



Fonte: Autoria própria.

Detalhadamente, as etapas do funcionamento da aplicação podem ser descritas da seguinte forma:

- Ler os Dados: após a inicialização, o usuário seleciona o arquivo com os dados de uma proteína para jogar e, na sequência, o arquivo é carregado;
- Gerar a Estrutura 3D: os dados da estrutura são utilizados para inicializar as posições dos objetos jogáveis;
- Representar a Estrutura em RV: como o motor gráfico (Unity) está configurado para uma aplicação em RV, esta etapa consiste na aplicação da renderização gráfica da etapa anterior;
- Receber Comandos do Joystick: o jogador, então, visualiza a proteína e utilizando o *joystick*, seleciona o aminoácido que deseja movimentar, alterando a estrutura por meio do *thumbstick* (eixos X e Y) e dos botões frontais (eixo Z);
- Alterar a Estrutura 3D: a posição do resíduo selecionado sofre uma variação na direção e sentido escolhido;

- Calcular a Função de Energia: a energia potencial é recalculada utilizando a função de energia do modelo 3D-AB *off-lattice* (Equação 1);
- Calcular a Pontuação: por sua vez, a pontuação é estabelecida conforme a variação da energia em relação ao seu valor inicial (ver Equação 6);
- Gravar os Dados: após a estabilização da estrutura, por meio do motor de física, a posição atual dos objetos que a compõe é salva dinamicamente para a operação de Desfazer/Refazer e assim a estrutura continua sendo disponibilizada ao usuário para novas iterações, até o encerramento da aplicação por meio dos menus do jogo.

3.3 PLATAFORMA DE DESENVOLVIMENTO

Uma vez que a ferramenta apresenta semelhança com jogos virtuais, optou-se por desenvolver a aplicação numa plataforma desse nicho. Assim, foi escolhida a Unity[1] (versão 2019.2), uma plataforma de desenvolvimento de jogos que conta com: suporte para aplicações em RV, incluindo o Google Cardboard[2], disponibiliza versões gratuitas e atualizadas para desenvolvedores com baixo orçamento, possui amplo suporte para o aprendizado e uma comunidade ativa com muitos grupos de discussão, inclusive no desenvolvimento de aplicações para RV.

Tendo sido escolhida a Google Cardboard como plataforma de realidade virtual, foi necessário importar para a Unity um Kit de Desenvolvimento de Software (*Software Development Kit* - SDK), contendo as bibliotecas e *templates* próprios para a utilização da mesma na implementação das funcionalidades de RV. Assim, seguindo as instruções no site do Google VR[3], foi adicionado e configurado o pacote do Google Cardboard SDK à plataforma Unity.

Com relação à linguagem de programação escolhida, para dar suporte a aplicação, optou-se pelo C Sharp (C#). Isso devido a ser reconhecidamente empregado na plataforma. Ainda, como no editor da Unity o desenvolvimento se faz de modo exclusivamente gráfico, para a edição dos *scripts* o ambiente de desenvolvimento (IDE) *Microsoft Visual Studio Community* [4] (MSVC) (versão 2017) foi sincronizado com a Unity. O MSVC também é uma versão gratuita, o que auxiliou na redução dos custos do projeto, sem perda de qualidade e produtividade.

[1] https://unity.com/
[2] https://arvr.google.com/cardboard/
[3] https://developers.google.com/vr/develop/unity/get-started-android
[4] https://visualstudio.microsoft.com/pt-br/

Em relação aos *scripts* criados no projeto, todos foram comentados seguindo o padrão do C# para os cabeçalhos. Com isso, é possível visualizar, dentro do MVSC, a descrição do método ou classe apresentada no comentário. Ademais, há comentários seguindo as linhas de instruções, com a finalidade de facilitar a compreensão dos atributos e métodos das classes.

Para realizar o controle de versões, o sistema de *backup* e o compartilhamento de arquivos foi utilizada a plataforma GitHub[5]. Assim, além da maior segurança contra perda de dados e acessibilidade conferida ao projeto, também é possível que o código da aplicação fique disponibilizado, para desenvolvedores interessados no seu funcionamento ou em aprimorar a ferramenta independentemente. O link para acessar o repositório é: `https://github.com/lucasdf09/TCC_PROTEIN_FOLD_VR`.

3.4 INTERFACE DE REALIDADE VIRTUAL

A proposta de interação do usuário com a aplicação num ambiente de RV demandou a utilização de um smartphone com SO Android, óculos de RV e um controle manual.

Para tanto, o smartphone utilizado teve de ser compatível com a aplicação Google Cardboard, disponível na loja Google Play, e rodar um SO Android 4.4 KitKat (API *level* 19) ou superior. Desse modo, o aparelho utilizado foi um Samsung Galaxy S8 Plus operando com o sistema Android 9.0 Pie. A função do smartphone na interface RV é a de processar a aplicação, atuar como *display* para a imagem estereoscópica 3D, conectar-se ao controle e informar a movimentação de cabeça do usuário, por meio de sensores como o giroscópio (Unity, 2020).

Além do smartphone, se fez necessário o uso de Óculos de RV e controle manual, do tipo *joystick*, para implementar a interação do usuário com a aplicação. Assim, optou-se por adquirir um kit que continha ambos os componentes. Denominado VR BOX, o conjunto é composto por um óculos de RV, consistindo em óculos com um par de lentes ajustáveis, ao qual o smartphone é acoplado, e um controle *bluetooth*, que possui um *thumbstick* e oito botões, sendo esse alimentado por duas pilhas palito (AAA). As Figuras 12 e 13 ilustram esses componentes.

[5]https://github.com/

Figura 12: Óculos de RV - VR BOX: (a) vista frontal e (b) vista traseira.

(a) (b)

Fonte: (a) e (b) Autoria própria.

Figura 13: Controle *bluetooth* - VR BOX: (a) vista frontal e (b) vista lateral.

(a) (b)

Fonte: (a) e (b) Autoria própria.

O controle que acompanha o kit VR BOX apresenta quatro funções de operação (Modo, Música, Vídeo e Jogos) e cada função apresenta um mapeamento botão/ação diferente. Para selecionar uma função, deve-se manter pressionado o botão arroba (@) simultaneamente com um dos botões A, B, C ou D por três segundos. Assim é selecionada a função Música, Jogos, Vídeos ou Modo, respectivamente. Para a aplicação, após análise da resposta dos botões, foi escolhida a função Jogos (@ + B) como base para o mapeamento dos botões, portanto, esta deve ser selecionada pelo usuário para a utilização do aplicativo. Na Unity, o mapeamento foi realizado pressionando cada um dos botões e lendo a saída gerada por meio da função nativa Input.GetKey (Unity, 2019j). O Quadro 3 apresenta a relação obtida para a função Jogos.

Quadro 3: Relação entre os botões do *joystick* e os valores obtidos em Input.GetKey.

Botão	Input.GetKey
Thumbstick	Cima, baixo, esquerda, direita
Botão de ligar	Não detectado
@	Não detectado
A	ALT direito
B	Botão 0 do *joystick*
C	ALT esquerdo
D	Botão 1 do *joystick*
Frontal superior	SHIFT direito
Frontal inferior	SHIFT esquerdo

Fonte: Autoria própria

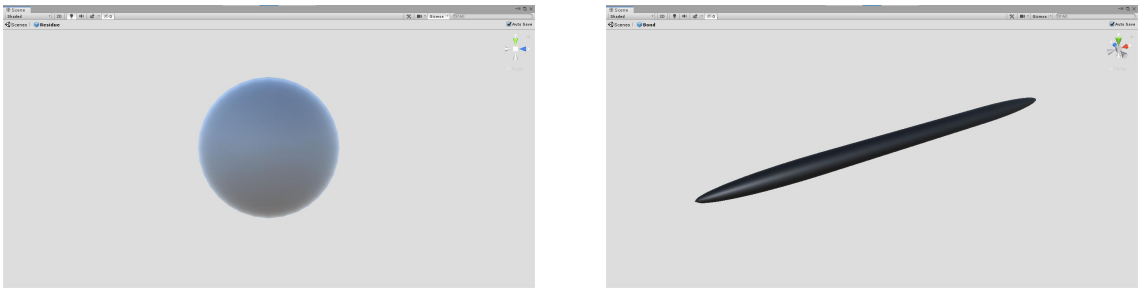
3.5 IMPLEMENTAÇÃO DO MODELO

A representação gráfica da estrutura proteica para visualização e manipulação utilizou formas geométricas tridimensionais simples. Isso, porque, o modelo 3D-AB *off-lattice* é um modelo *coarse-grained*, no qual os aminoácidos, suas ligações e o meio no qual a cadeia está imersa, são simplificados. Assim, a partir de formas preexistentes de esfera e cápsula, foram criados dois *templates* de objetos (denominados Prefabs dentro do editor da Unity) (Unity, 2019d).

Com base na esfera foi criado o prefab Residue (resíduo), para representar um aminoácido da cadeia. Por sua vez, o prefab Bond (ligação) foi criado tendo por base uma cápsula e representa a ligação entre dois aminoácidos. A Figura 14 apresenta esses elementos.

Figura 14: Prefabs: (a) Residue e (b) Bond.

(a) (b)

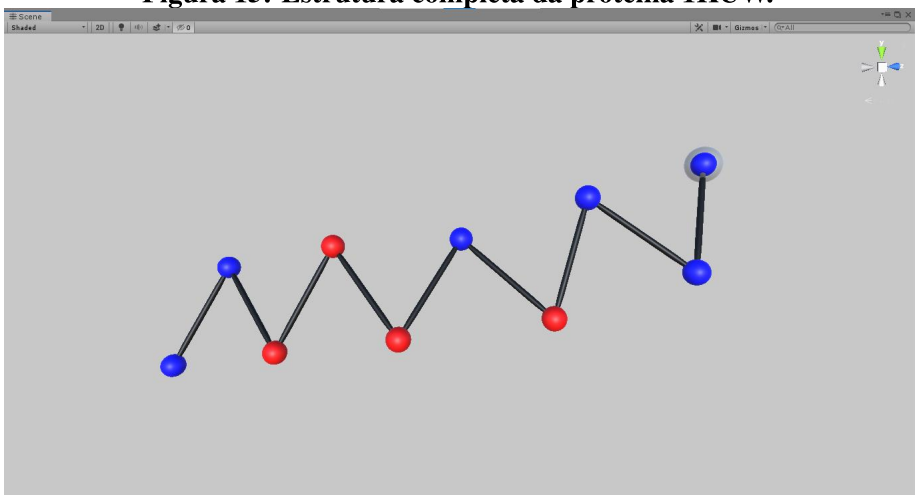


Fonte: Autoria própria.

A fim de utilizar o motor de física da Unity para controlar a movimentação dos resíduos e colisão entre os elementos, foi adicionado o componente Rigidbody aos prefabs Residue e Bond. Os componentes são classes que determinam o comportamento do objeto que os contém e o componente Rigidbody, por sua vez, atua no controle da posição de um objeto por meio de simulação (motor de física), não sendo necessários *scripts* adicionais para gerenciar a colisão e movimentação coesa dos elementos da estrutura (Unity, 2019e, 2019g).

Outro componente importante para a construção da estrutura de visualização e manipulação foi a Joint. Esse componente conecta dois objetos que contém um Rigidbody ou um objeto com Rigidbody a um ponto fixo no espaço (Unity, 2019f). Com isso, foram utilizados dois tipos de Joints: a Fixed Joint e a Configurable Joint, ambas adicionadas ao prefab Bond. A Fixed Joint de um objeto Bond foi utilizada para uni-lo ao seu Residue contíguo subsequente, imobilizando esse resíduo na extremidade da ligação. Com relação às Configurable Joints, essas foram utilizadas para conectar um Bond ao seu contíguo precedente, conferindo mobilidade angular em todas as direções (*yaw*, *pitch*, e *roll*) para um Bond em relação ao seu vizinho. No caso do primeiro Bond, sua Configurable Joint foi utilizada para fixá-lo ao primeiro Residue e configurada para responder da mesma forma que uma Fixed Joint. A Figura 15 apresenta o resultado da estrutura completa.

Figura 15: Estrutura completa da proteína 1KUW.



Fonte: Autoria própria.

Por fim, para o cálculo da pontuação, foi utilizada a função de energia do modelo 3D-AB *off-lattice*, apresentada na Equação 1. Desse modo, ao início de um novo jogo, é calculada a energia potencial da estrutura e conforme o usuário altera a posição dos resíduos, a energia potencial da nova estrutura é recalculada e a pontuação é dada como a diferença entre a energia potencial computada no início e a atual, como está simplificado na Equação 6.

$$Pontuação = E_{P_{inicial}} - E_{P_{atual}} \quad (6)$$

Assim, uma pontuação positiva significa uma energia potencial da estrutura atual ($E_{P_{atual}}$) menor que a energia inicial ($E_{P_{inicial}}$), indicando maior estabilidade do ponto de vista energético. Por sua vez, uma pontuação negativa sinaliza uma estrutura com maior energia interna e menor estabilidade, quando comparada à estrutura inicial. Com isso, a coloração do placar se altera para indicar ao usuário seu desempenho. A cor preta indica pontuação nula ou zero, a cor verde uma pontuação positiva e a cor laranja uma pontuação negativa.

3.6 CONVERSÃO DE FORMATOS

As estruturas de proteínas utilizadas como entrada, para iniciar uma nova sessão na aplicação, um novo jogo, precisam estar num formato específico. Tendo em vista a colaboração com o LABIC, esse formato segue o padrão de saída de simulações executadas no laboratório, utilizando o algoritmo de DM publicado em (BENÍTEZ; LOPES, 2012; BENÍTEZ; LOPES, 2013). Esse algoritmo toma como argumento de entrada, dentre outros parâmetros, a sequência de aminoácidos de uma proteína globular, codificados segundo o Quadro 2. A saída, por sua vez, consiste em arquivos que apresentam dados relativos à simulação, tais como: a posição dos resíduos dentro de um intervalo periódico, os parâmetros de análise da estrutura ao longo do processo e a melhor estrutura obtida.

Desse modo, o arquivo utilizado na aplicação é essa saída com a melhor estrutura. Ele apresenta a posição cartesiana, no espaço, dos resíduos da sequência de aminoácidos, com a menor energia interna encontrada por meio da simulação de DM. Para ser utilizado como entrada, esse arquivo deve estar no formato texto (`.txt`) e conter, ao menos: as posições dos aminoácidos, a sua sequência no formato AB e a quantidade dos mesmos. Um exemplo de arquivo de entrada, utilizado no jogo, está disponível no Anexo A.

Com relação aos arquivos de entrada *default* da aplicação, esses são cinco e ficam armazenados no diretório /_GameFiles/Inputs, sendo eles: 10_1KUW (contendo 10 resíduos), 13_FIBO (13 resíduos), 13_teste (13 resíduos), 56_1NLO (56 resíduos) e 104_1HJM (104 resíduos). Desse modo, a conversão desses arquivos num objeto jogável se dá pela sua leitura e identificação da quantidade, tipo (A ou B) e posições dos resíduos, sendo, então, realizada a atribuição desses valores aos objetos de jogo. O resultado pode ser visualizado na Figura 15, na qual as esferas são instâncias do prefab Residue, ou seja, os aminoácidos, e as hastes escuras que ligam um resíduo a outro são instâncias do prefab Bond. Sendo que o primeiro elemento

da sequência está envolto por uma esfera translúcida. Quanto a coloração da esfera, azul indica que o aminoácido é do tipo B (hidrofílico) e vermelho que é do tipo A (hidrofóbico), em acordo com o Quadro 2.

Por fim, os arquivos de entrada *default* tiveram de ser inseridos dentro do pacote de instalação, sendo posteriormente transferidos para o local de armazenamento do aplicativo no sistema. Para isso, os arquivos foram alocados no diretório StreamingAssets. Nesse diretório, os arquivos persistem no pacote de instalação e são acessados por meio da constante nativa Application.streamingAssetsPath (Unity, 2019i). Nos sistemas Android, a transferência de arquivos da pasta StreamingAssets é feita somente por requisição web (semelhante à comunicação com um servidor web) (Unity, 2019i). Assim, foi utilizada a classe nativa UnityWebRequest na transferência dos arquivos em StreamingAssets para /_GameFiles/Inputs, um diretório dentro do sistema. E, com isso, após a primeira inicialização do aplicativo, ocorre apenas uma verificação dos arquivos. Feita por meio da comparação dos nomes dos arquivos no sistema com uma lista das entradas *default*. Dessa maneira, somente em caso de falha ao encontrar os arquivos *default* é realizado um novo acesso ao diretório StreamingAssets.

3.7 INTERAÇÃO DO USUÁRIO

O uso de RV necessita, ao mesmo tempo em que possibilita, diferentes formas de o usuário interagir com o conteúdo virtual. Além disso, a usabilidade da aplicação explora o conceito de gamificação, tanto para simplificar a compreensão da proposta quanto para atrair a atenção do usuário. Portanto, o método de *Gaze Input*, um controle *bluetooth* e um teclado virtual foram utilizados para prover ao usuário meios de interação.

Primeiramente, o conceito de *Gaze Input*, que pode ser traduzido do inglês como “entrada pelo olhar”, emula o uso do mouse nos sistemas operacionais com interface gráfica. Na aplicação, consiste em um cursor circular localizado no centro do campo de visão (GoogleVR, 2019). Assim, ao mover a cabeça, o usuário pode posicionar o cursor sobre objetos interativos, como botões de menu e componentes jogáveis, e selecioná-los utilizando o botão C do *joystick* (mapeado como “C”, de acordo com o Quadro 4). A implementação do *Gaze Input* foi realizada utilizando o pacote de desenvolvimento Google VR Unity e a lógica de implementação do clique está descrita na Figura 33 no Anexo C.

Para realizar as funções de acesso direto aos menus e movimentação espacial, foi utilizado um controle com conexão *bluetooth*. Esse controle, que pode ser visto na Figura 13, possui seis botões mapeados para a aplicação, os botões superiores A, B, C e D, mais dois botões frontais na forma de gatilho, além de um *thumbstick*, utilizado para realizar as entradas das setas do teclado (vertical e horizontal).

Desse modo, utilizando a informação do Quadro 3, foram criadas novas entradas de botões para gerenciar a leitura do *joystick*. E assim, dentro da aplicação, a leitura das entradas do controle é realizada via *script* pelos métodos nativos Input.GetButton para os botões e Input.GetAxis / Input.GetAxisRaw para o *thumbstick* (Unity, 2019j).

Quadro 4: Relação entre os botões do *joystick* (Input.GetKey) mapeadas para Input.GetButton.

Input.GetKey	Input.GetButton
Eixo Y do *thumbstick*	Horizontal
Eixo X do *thumbstick*	Vertical
SHIFT esquerdo	Z-axis (negativo)
SHIFT direito	Z-axis (positivo)
ALT direito	A
Botão 0 do *joystick*	B
ALT esquerdo	C
Botão 1 do *joystick*	D
SHIFT direito	Fire1
SHIFT esquerdo	Fire2

Fonte: Autoria própria

Ainda, a necessidade de uma entrada de texto levou à implementação de um teclado virtual para ser utilizado dentro do ambiente de RV. A partir do *layout* QWERTY, foram criados três painéis com botões para digitação, apresentados na Figura 16. O painel StandardKeyboard é o primeiro a ser mostrado e implementa as letras minúsculas do alfabeto latino (a - z). Além dessas, há cinco teclas de função, na faixa inferior, que são comuns a todos os painéis. São elas: Back - apaga o último caractere digitado; Enter - conclui a rotina de entrada e Space - insere um caractere de espaço em branco. Os outros dois botões variam de acordo com o teclado selecionado, tendo a função de chave seletora entre os tipos: padrão ou minúscula (abc), maiúscula (ABC) e numérico simbólico (123). O painel CapitalKeyboard implementa as letras maiúsculas (A - Z). E o painel NumericalKeyboard apresenta as entradas dos dez algarismos indo-arábicos (0 - 9) e nove entradas simbólicas, sendo elas: @ , # , $, + , = , - , _ , (,) .

Figura 16: Teclado virtual: (a) StandardKeyboard, (b) CapitalKeyboard e (c) NumericalKeyboard.

(a)

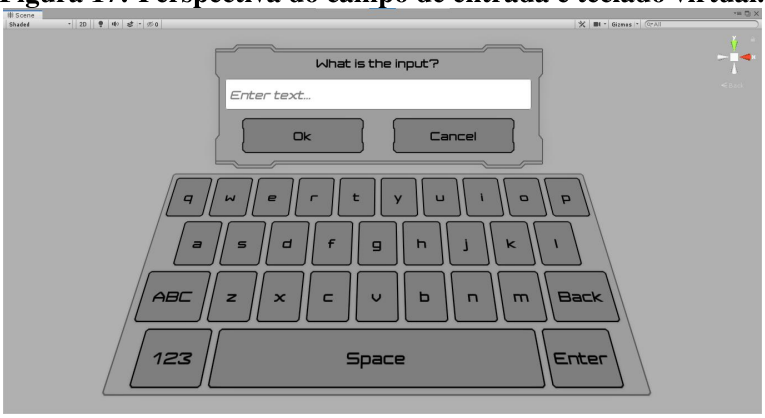
(b)

(c)

Fonte: Autoria própria.

Outro elemento necessário à entrada de texto foi um campo para visualização dos caracteres digitados. Essa funcionalidade foi implementada utilizando a classe InputField (Unity, 2019a). Assim, a digitação no teclado virtual funciona com o posicionamento do *pointer* sobre a tecla desejada e pressionamento do botão de acionamento do *joystick* (C), semelhante ao uso dos botões de menu. A Figura 36 no anexo D apresenta os métodos criados para essa função, bem como a organização dos elementos que compõem o teclado virtual. Ainda, a Figura 17 ilustra a perspectiva desses elementos pelo usuário dentro da aplicação.

Figura 17: Perspectiva do campo de entrada e teclado virtual.



Fonte: Autoria própria.

4 RESULTADOS E DISCUSSÕES

O presente capítulo descreve os aspectos finais da aplicação, os testes realizados e as conclusões sobre o resultado alcançado.

4.1 APLICAÇÃO FINAL

O resultado final obtido configura um aplicativo com uma interface característica de jogos virtuais, mas, com uma temática voltada para um problema de interesse da ciência. Dentre os elementos que descrevem o aspecto e funcionamento da aplicação estão os métodos de inicialização, o que ocorre dentro do jogo e os menus.

4.1.1 INICIALIZAÇÃO

Antes de iniciar a aplicação, o usuário deve conectar o controle *bluetooth* ao smartphone, no modo Jogo. Em seguida, abrir o aplicativo, acomodar o aparelho nos óculos de RV e vesti-los na cabeça. Na sequência, é executada uma rotina de verificação para obter os *joysticks* conectados ao aparelho, utilizando o método nativo Input.GetJoystickNames, que retorna uma lista com os mesmos (Unity, 2019k). Assim, se o tamanho da lista é maior que zero e o conteúdo não é nulo, é apresentado o menu principal (Main Menu). Caso contrário, o painel de menu é desabilitado e uma janela de notificação solicita ao jogador que conecte o controle. Como o botão de confirmação é acionado utilizando o controle, o jogo só continua após esse requisito ser cumprido.

No mais, a fim de gerenciar a inicialização da aplicação, foi implementada a classe GameFilesHandler. Essa classe centraliza todos os métodos de acesso, leitura e modificação de arquivos e diretórios dentro da aplicação. E, como esse objeto precisa persistir durante toda a execução, inclusive entre as Scenes (elemento da Unity que contém os objetos de um ambiente de jogo (Unity, 2019h)) ele foi implementado como um *singleton*, de modo que não pode ser instanciado mais de uma vez.

Ainda, durante a inicialização, notou-se que ocorriam mudanças muito pequenas no valor da pontuação. Para contornar esse problema, no início de uma sessão, foi implementada uma rotina de estabilização dos objetos Residues e Bonds, tanto para um novo jogo quanto para um jogo salvo. Essa rotina reduziu as alterações na posição desses objetos de jogo, durante a primeira ação do motor de física, utilizando os elementos Rigidbody.isKinematic (Unity, 2019o) e Physics.autoSimulation (Unity, 2019m). Um diagrama ilustrando as etapas do processo pode ser consultado na Figura 34, no Anexo C. Resumidamente, com essa abordagem, as variações observadas foram da ordem de 10^{-7} em testes realizados nas entradas *default*. Exemplificando, a estrutura 10_1KUW apresentou uma variação inicial na pontuação de $9,536743 * 10^{-7}$.

No entanto, como mostra a Equação 6, isso apresenta um resultado incomum ao usuário, uma vez que ao iniciar sua jogada o placar não está zerado. Com isso, após a rotina de estabilização, a energia interna calculada é utilizada como referência para a pontuação, garantindo um valor inicial nulo. Além disso, em testes realizados, constatou-se que o tempo para a realização desse procedimento não é perceptível ao usuário, mesmo em estruturas maiores como a 104_1HJM, que possui mais de cem resíduos.

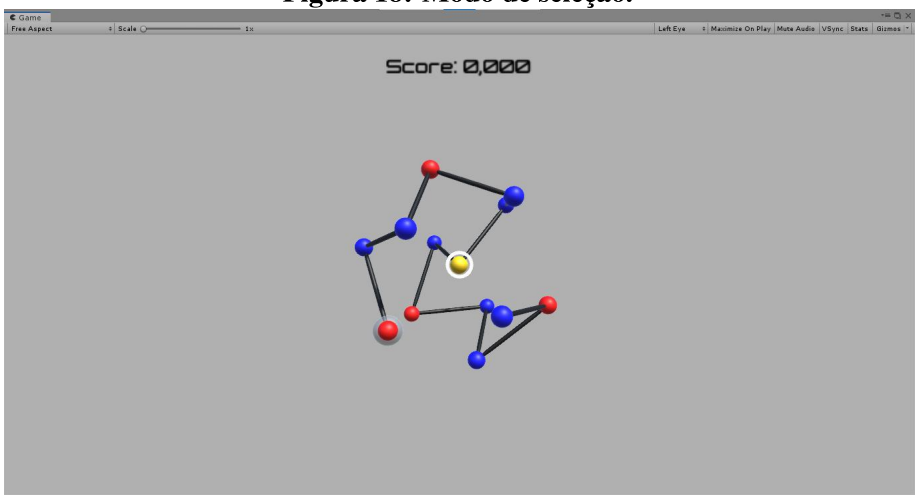
4.1.2 DENTRO DO JOGO

Uma vez que o usuário selecionou e confirmou a estrutura com a qual ele deseja jogar, ocorre a transição de Scene para o ambiente de jogo. A Figura 41 no Anexo D apresenta os elementos que compõem esse ambiente. Ainda, a fim de descrever a usabilidade da aplicação, foi elaborada, no Anexo B, uma sequência de imagens e há também um vídeo[1], na plataforma YouTube, que demostra brevemente essa jogabilidade.

A primeira ação a ser realizada na rotina de jogo é selecionar o aminoácido que se deseja movimentar. Para isso, deve-se posicionar o Reticle Pointer sobre a esfera, que começará a variar entre sua cor original e a cor amarela. Ao clicar com o botão C do *joystick*, a aplicação entrará em modo de movimento e a esfera passará a variar sua transparência entre a cor original opaca e um tom semitransparente. Ao sair do modo de movimento e retornar ao modo de seleção, a coloração do objeto retornará ao seu aspecto inicial. A Figura 18 apresenta a mudança de cor da estrutura no modo de seleção.

[1] https://youtu.be/OeKYdKeXQ2Q

Figura 18: Modo de seleção.

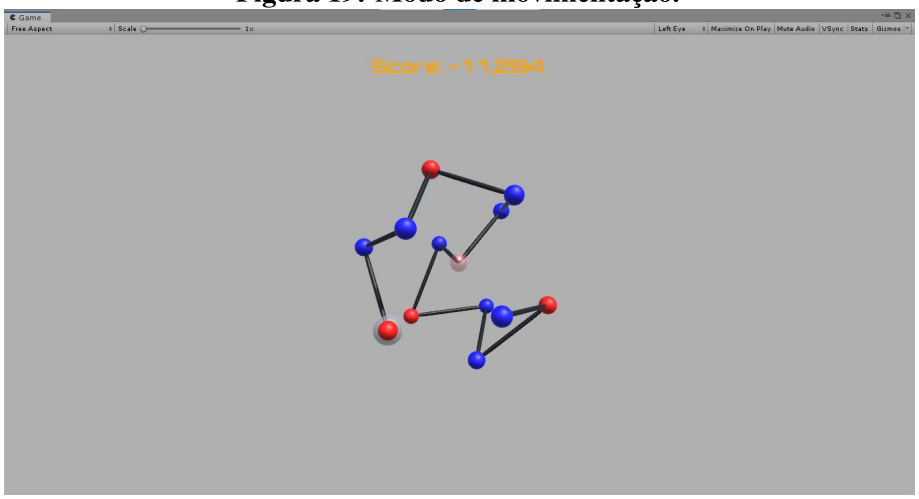


Fonte: Autoria própria.

Em relação à manipulação dos aminoácidos, essa é gerenciada pela classe PlayerControler. Nessa classe encontram-se os métodos que averíguam as entradas do *thumbstick* para movimentos horizontais (no eixo x) e verticais (no eixo y). Os movimentos no eixo z ficam por conta dos botões frontais superior (sentido positivo) e inferior (sentido negativo). Por meio do método nativo FixedUpdate, é possível realizar a execução periódica de um conjunto de instruções (Unity, 2019l). Assim, após a seleção de um resíduo, a *flag* move_mode é acionada e a aferição do controle *bluetooth* e aplicação do deslocamento são executadas.

Para checar as entradas é utilizado o método Input.GetAxisRaw e em caso de detecção é aplicada uma translação de valor fixo. Dessa maneira, primeiramente é calculada a nova posição e, na sequência, é aplicado o deslocamento na coordenada espacial do resíduo pelo método Transform.Translate (Unity, 2019r). Isso, porque esse método permite tomar como referencial de deslocamento a posição da câmera. A Figura 19 ilustra a mudança que ocorre, durante o modo de movimentação, na cor na partícula e da pontuação.

Figura 19: Modo de movimentação.



Fonte: Autoria própria.

Enquanto está no modo de seleção, dentro do jogo, o usuário pode modificar a sua perspectiva da estrutura, por meio do *joystick*. O movimento de revolução utiliza o método nativo RotateAround() (Unity, 2019q) para alterar a posição e rotação do objeto Player, dando a impressão de que é a estrutura que está se movimentando. A revolução é feita utilizando o *thumbstick* e o *zoom* utiliza o botão frontal superior para aproximar e o inferior para afastar, por meio de Transform.Translate. Essa técnica foi escolhida porque não modifica a posição dos resíduos, o que seria mais custoso e poderia gerar erros devido ao acumulo de arredondamentos durante as transposições.

O ajuste do posicionamento da câmera utiliza o centro de massa da estrutura. Assim, como cada resíduo possui massa unitária, essa coordenada coincide com o centro geométrico (HALLIDAY et al., 2012). Esse ponto foi utilizado porque, empiricamente, apresentou uma perspectiva equidistante de todas as extremidades da molécula. A Equação 7 traz a expressão geral para n partículas, com massa m_i, posicionadas ao longo do eixo x, utilizada para calcular o centro de massa da estrutura (HALLIDAY et al., 2012).

$$x_{CM} = \frac{m_1x_1 + m_2x_2 + m_3x_3 + ... + m_nx_n}{m_1 + m_2 + m_3 + ... + m_n} = \frac{1}{M}\sum_{i=1}^{n} m_ix_i \tag{7}$$

Desse modo, a partir da Equação 7, estendendo o cálculo para todas as coordenadas espaciais (x, y, z) dos resíduos $(residues[k])$ e considerando que a massa total do sistema é a soma da massa unitária dos resíduos $(M = n\_mol)$, o ponto de centro de massa (p_{CM}) da estrutura proteica foi calculado conforme a Equação 8.

$$p_{CM}(x,y,z) = \frac{1}{n\_mol} \sum_{k=0}^{n\_mol-1} residues[k](x,y,z) \quad (8)$$

Com isso, foi estabelecido um método para determinar a posição inicial do jogador, de modo a acomodar a estrutura da proteína dentro do seu campo de visão. Primeiro calcula-se o resíduo mais próximo e o mais distante do centro de massa. Em seguida, computa-se o produto externo entre os resíduos para se obter um vetor perpendicular. Na sequência, esse vetor é normalizado e multiplicado pelo dobro da distância do centro de massa até o resíduo mais distante. Então, o Player é posicionado na coordenada obtida e o ajuste da câmera feito por meio do método nativo LookAt, que centraliza o campo de visão na coordenada desejada, no caso, o centro de massa (Unity, 2019p).

A aplicação também conta com a operação de desfazer e refazer jogadas. Para isso, dentro da classe PlayerController foi implementada uma lista (undo_list) com instâncias da classe PlayState, construída para armazenar posições e rotações de aminoácidos e ligações, e a rotina de desfazer movimentos, que atua em conjunto com os processos de movimentação. O diagrama da Figura 35, no Anexo C, apresenta a sequência de procedimentos que descreve essa funcionalidade. Vale destacar, ainda, que a lista é alocada estaticamente, possuindo 1048576 (2^{20}) posições, o que torna seu acesso otimizado, uma vez que é muito frequente durante o tempo de execução.

4.1.2.1 ARQUIVOS DE JOGO

Na aplicação há finalidades que envolvem a manipulação de arquivos, como: salvamento de um jogo, carregamento de jogos salvos, geração de saídas exportáveis e inclusão de novas estruturas iniciais.

O salvamento de uma sessão ocorre dentro do ambiente de jogo. O usuário seleciona no menu de jogo a opção salvar (Save). Em seguida, o teclado virtual é apresentado e o jogador deve inserir um nome para o arquivo a ser salvo. Então, é realizada a verificação da *string* de entrada, de maneira que o nome digitado seja rejeitado em caso de: entrada em branco (ou nula), entrada iniciada com espaço em branco, entrada finalizada com espaço em branco ou nome já existente. Em caso de entrada inválida, é apresentada uma janela de diálogo informando o problema. Para as entradas válidas, os dados de recuperação são salvos dentro de uma instância da classe de salvamento. E, ainda, há um atributo em GameFilesHandler que limita a quantidade de salvamentos.

Desse modo, a classe SaveData foi construída, com atributos para armazenar: o nome do arquivo de salvamento, o nome do arquivo original, o número de partículas, a sequência AB, a melhor energia do arquivo original, a pontuação, as posições e rotações dos objetos que representam os resíduos e ligações e a posição e rotação da câmera. Por meio, então, de uma função de serialização nativa da Unity, esses dados são salvos no formato JSON (*JavaScript Object Notation*) dentro do diretório /_GameFiles/Saves. Além disso, como esses dados não são sensíveis, não foi utilizada uma política de segurança, somente sendo assegurada a integridade dos mesmos.

O procedimento para o carregamento de um jogo salvo segue a lógica inversa. No menu principal, o usuário opta por carregar um jogo salvo de uma lista que lhe é apresentada. Após selecionar o botão com o nome da sessão armazenada, os dados salvos são recuperados de um arquivo JSON e inicializados pelo mesmo procedimento (*script*) de um novo jogo.

Outro ponto explorado nessa rotina de salvamento foi a simultânea geração de um arquivo de saída. Com isso, para toda estrutura salva, existe um arquivo texto no diretório /_GameFiles/Outputs, formatado de maneira semelhante aos arquivos de entrada.

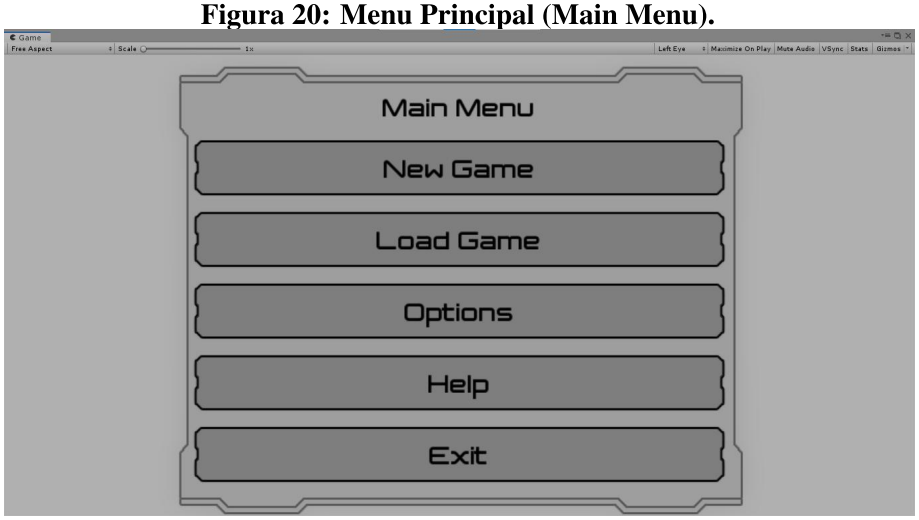
Além disso, a adição de novas estruturas é possível pela inclusão do respectivo arquivo no formato adequado, como no Anexo A. Assim, selecionado o arquivo a ser adicionado à lista de novos jogos, deve-se acessar o diretório /_GameFiles/Inputs, por meio do explorador de arquivos do smartphone, e realizar a inserção. Por exemplo, no Samsung Galaxy S8 Plus, o caminho acessado pelo SO Windows 10 foi: Este Computador\Galaxy S8+\Phone\Android\data\com.TCCUTFPR.ProteinFoldVR\files_GameFiles\Inputs.

Enfim, para evitar que uma estrutura incompatível com o modelo seja carregada, é realizada uma verificação antes da inicialização. Após a escolha de uma proteína na lista de novo jogo, são testadas as seguintes condições: o número de resíduos é igual ao número de coordenadas, há apenas os caracteres A e B na sequência e eles correspondem à quantidade de resíduos, os formatos das entradas (*int* para o número de resíduos, *char* para a sequência e *float* para as coordenadas) e por fim, é averiguada a distância unitária entre os resíduos contíguos. Se o conteúdo do arquivo corresponder às especificações, ele é carregado normalmente. Caso contrário, é apresentado um painel de notificação.

4.1.3 MENUS

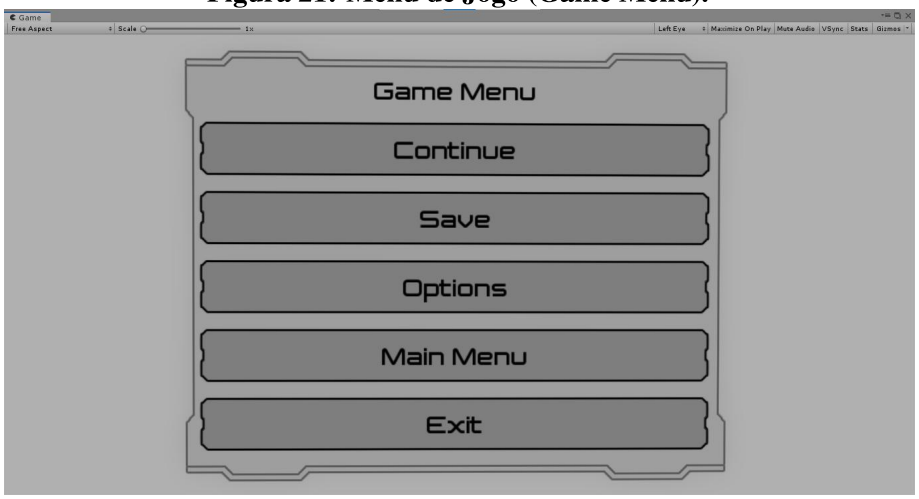
A aplicação encontra-se organizada de tal maneira que as funcionalidades não relacionadas à visualização e manipulação da proteína estão centralizadas em dois elementos de menu. Na tela inicial, tem-se o menu principal (Main Menu), ilustrado pela Figura 20, no qual o usuário tem a opção de: escolher um novo jogo, carregar um jogo salvo, ajustar a orientação do campo de visão, acessar os tutoriais e sair do jogo. O outro está no ambiente de jogo, sendo esse o menu de jogo (Game Menu), podendo ser visualizado na Figura 21, nele é possível: voltar ao jogo (continuar), salvar um novo jogo, sobrescrever o jogo atual, apagar um jogo salvo, mostrar informações da estrutura em execução, habilitar ou desabilitar e ajustar a posição dos mostradores de pontuação (Score) e parâmetros (Parameters), ajustar a orientação do campo de visão, acessar os tutoriais, retornar à tela inicial (menu principal) e sair do jogo.

Figura 20: Menu Principal (Main Menu).



Fonte: Autoria própria.

Figura 21: Menu de Jogo (Game Menu).



Fonte: Autoria própria.

Os menus do jogo foram desenvolvidos a partir de *layouts* presentes no Asset Unity Samples: UI[2], disponibilizado gratuitamente na loja virtual Unity Store. Assets são representações de itens que podem ser utilizados no projeto, um Asset pode vir de um arquivo externo ao ambiente Unity, como um modelo 3D, um arquivo de áudio ou uma imagem (Unity, 2018). Sendo assim, partindo do *template* de objeto Panel, foi criado um prefab para os menus denominado MenuPanel. A Figura 37 no Anexo D apresenta a estrutura conceitual desse prefab, assim como do MenuButton. De modo que ambos foram utilizados em conjunto para implementar diferentes elementos de menu.

Os menus em formato de lista são utilizados em operações cuja quantidade de objetos a serem escolhidos varia, por exemplo, na seleção dos arquivos de jogos salvos. Tendo isso em mente, foi criado um *template* desse tipo de objeto, para que fosse reaproveitado mais facilmente. Por isso, foram criados os prefabs ListPanel e ListButton, utilizados na implementação dos menus de: novo jogo (Figura 22), jogo salvo, tutoriais e remoção de jogo salvo. Um diagrama representando a implementação desses prefabs é apresentado na Figura 38 do Anexo D.

[2]https://assetstore.unity.com/packages/essentials/unity-samples-ui-25468

Figura 22: Menu de seleção de novo jogo. Uma instância do prefab ListPanel.

Fonte: Autoria própria.

A transição entre o menu principal e o ambiente de jogo é feita por meio de uma mudança de Scene. No projeto são duas: a Main Scene e a Game Scene. No entanto, como não é possível acessar objetos de outra Scene, foram utilizados PlayerPrefs para referenciar a estrutura de novo jogo ou jogo salvo. Essa funcionalidade nativa destina-se a auxiliar os desenvolvedores no fluxo de dados simples (*ints*, *floats* e *strings*) entre as sessões de um jogo, por meio de uma relação chave-valor (Unity, 2019n). Sendo assim, na Main Scene, ao selecionar uma estrutura, seu caminho de acesso é armazenado na PlayerPref new_game, para um novo jogo ou saved_game, para um jogo salvo. E depois, na Game Scene, essa informação é recuperada, sendo identificada a sessão como de um novo jogo ou jogo salvo pelo valor da tupla não nula.

4.1.3.1 JANELA MODAL

Uma função muito utilizada na aplicação é a janela modal. Por meio dela é apresentado ao usuário a possibilidade de confirmar uma ação, evitando que cliques desprevenidos gerem alterações irreversíveis. Um exemplo é a janela de confirmação ao se acionar o botão Sair, informando que, caso o usuário prossiga (clicar em Ok), seu progresso será perdido. A Figura 23 apresenta essa situação.

Figura 23: Aplicação da funcionalidade Janela Modal no pedido de confirmação de saída da aplicação.

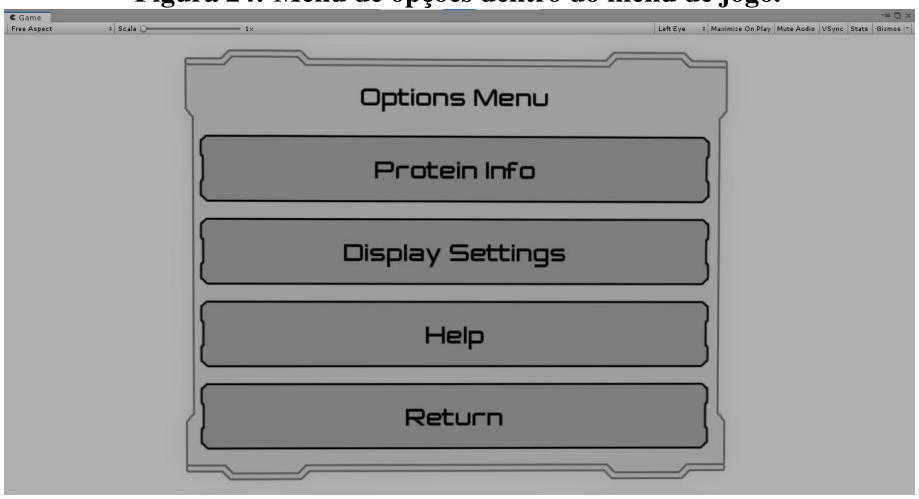
Fonte: Autoria própria.

Para essa funcionalidade foram criados dois *templates*, os prefabs ModalPanel e ModalButton, cuja implementação está representada na Figura 39 do Anexo D. A integração entre a interface e a ação é realizada por meio da classe ModalPanel. Essa classe permite que métodos em outros *scripts*, previamente referenciados, sejam invocados no evento de clique. Com isso, foi estabelecido que esses métodos estariam localizados nos botões que realizam a chamada da janela modal. E, por fim, como o processo de referenciação ocorre de maneira dinâmica, a instância de janela modal foi reaproveitada dentro de cada Scene.

4.1.3.2 OPÇÕES DO DISPLAY

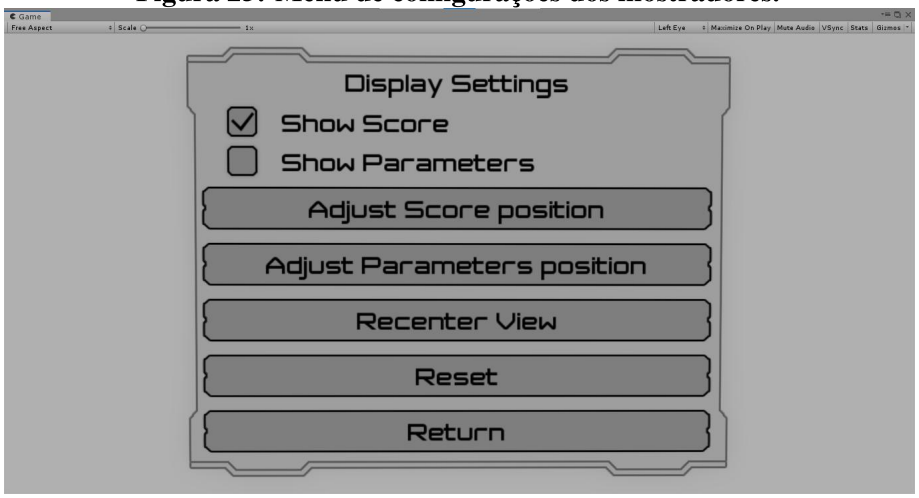
Os mostradores do jogo indicam ao usuário as informações de pontuação (Score) e parâmetros (Parameters) técnicos da estrutura, oriundos do algoritmo de DM, sendo esses o raio de giração total (Rg), o raio de giração dos resíduos hidrofóbicos (RgH) e o raio de giração dos resíduos polares (RgP). Esses objetos estão aninhados à câmera e portanto permanecem sempre a uma distância fixa do olhar do jogador. Para ajustar esses elementos e também centralizar a orientação dos óculos de RV, foi implementado o menu Display Settings (Figura 25). Acessado em Options (Figura 24), no Game Menu (Figura 21).

Figura 24: Menu de opções dentro do menu de jogo.



Fonte: Autoria própria.

Figura 25: Menu de configurações dos mostradores.



Fonte: Autoria própria.

O *layout* do menu Display Settings apresenta duas caixas de seleção (Toogles) que permitem ao jogador ativar e desativar a visualização dos mostradores (Unity, 2019b). Há, ainda, dois botões de ajuste da posição dos mostradores no campo de visão do jogador. Para isso, foi utilizada uma Coroutine, função cuja execução pode ser adiada por um intervalo de tempo específico e retomada entre as atualizações de *frame*, até que uma condição de parada seja alcançada (Unity, 2019c). Nesse caso, uma coroutine verifica o pressionamento do botão de menu (botão D), encerrando o modo de ajuste, enquanto o usuário modifica a posição do mostrador utilizando o controle. O menu ainda conta com o botão Reset, que posiciona os mostradores em coordenadas predefinidas.

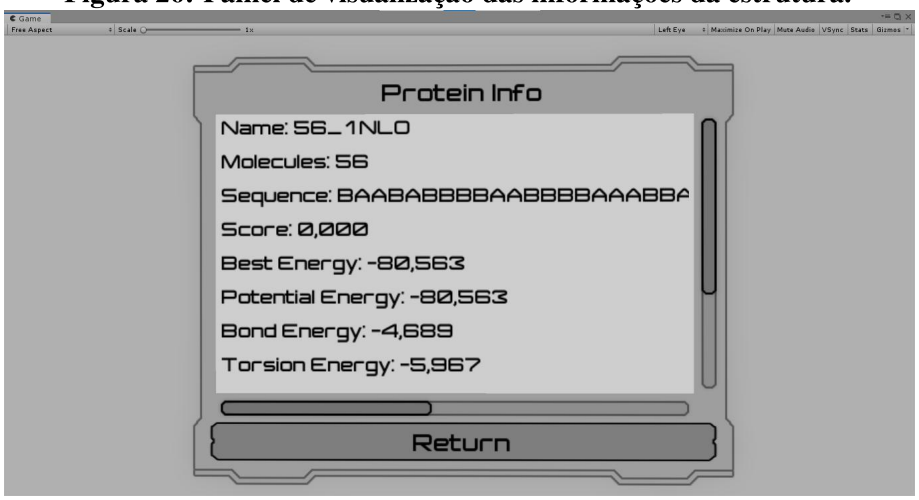
Com relação à opção de recentralização, esta encontra-se tanto no menu principal quanto no menu de jogo. A função é implementada por meio da coroutine receterMainView, que aguarda o pressionamento do botão D do *joystick* para ajustar o campo de visão do jogador. Dessa maneira, o método nativo UnityEngine.XR.InputTracking.Recenter é utilizado para reorientar a posição atual da cabeça do usuário (Unity, 2019s).

Por fim, há o botão de Return, que também permite o salvamento da posição dos mostradores. Para isso, foi implementada a classe SettingsData, utilizada em conjunto com o processo de serialização JSON. Assim, durante a inicialização, se o arquivo Display_settings for encontrado, seus dados são deserializados e as posições dos mostradores atualizadas.

4.1.3.3 INFORMAÇÕES DA PROTEÍNA

Permitindo ao usuário visualizar as informações da estrutura que está sendo processada, foi implementada a funcionalidade Protein Info, dentro das opções do menu de jogo, como mostra a Figura 26. Os dados apresentados são: nome, quantidade de moléculas, sequência AB, pontuação atual, melhor energia (referência para a pontuação), energia atual, parcelas que compõe a energia potencial (energia de ligação, energia de torção e potencial de Lennard-Jones), raio de giração geral, raio de giração hidrofóbico e raio de giração polar. Essas informações são carregadas quando o usuário ativa o botão Protein Info (Figura 24). Para mais detalhes da implementação do prefab InfoPanel, consultar a Figura 40 no Anexo D.

Figura 26: Painel de visualização das informações da estrutura.



Fonte: Autoria própria.

4.1.3.4 TUTORIAIS

Para instruir o usuário a utilizar a aplicação e sanar suas dúvidas durante a execução, foram elaborados dezesseis tutoriais. O botão de acesso (How to Play), situado dentro da seção de ajuda (Figura 27), carrega um menu de lista (Figura 28) no qual estão referenciados os arquivos de texto dentro da pasta /_GameFiles/Tutorials. Dessa maneira, após a seleção de um tutorial, a informação do arquivo é carregada no painel de leitura, ilustrado na Figura 29.

Há, ainda, a opção de visualizar informações sobre o jogo. Assim, ao clicar no botão About, o *script* LoadFileAbout utiliza o painel TutorialsInfoPanel para mostra o texto presente no arquivo About_Application, armazenado em /StreamingAssets/About.

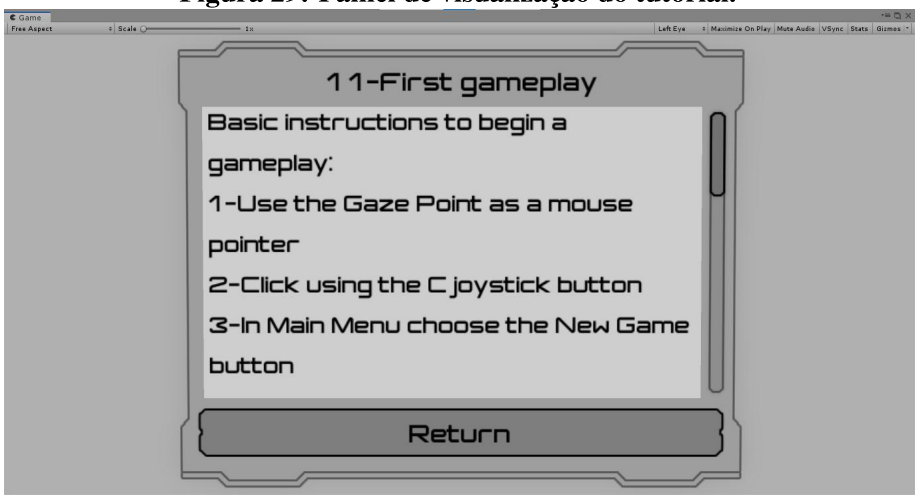
Figura 27: Menu de ajuda.

Fonte: Autoria própria.

Figura 28: Lista de tutoriais.

Fonte: Autoria própria.

Figura 29: Painel de visualização do tutorial.



Fonte: Autoria própria.

4.2 TESTES

Uma vez que a aplicação foi desenvolvida seguindo um modelo incremental, a cada nova funcionalidade implementada, eventos de teste foram realizados para sua validação. Desse modo, os testes foram realizados primeiramente no editor do *desktop* e em seguida no smartphone, conectado ao PC por um cabo USB, possibilitando, assim, a visualização das mensagens de *debug* durante a execução.

Ademais, após a conclusão do desenvolvimento, foi realizada uma bateria final de testes, que consistiu em executar a aplicação por trinta minutos ininterruptos, três vezes, com um intervalo irregular entre cada execução e sem a conexão USB. Essa rotina de testes teve a finalidade de avaliar aspectos como o conforto visual, o consumo de bateria, o aquecimento do aparelho, a presença de falhas e a usabilidade final da aplicação.

Sendo assim, durante a bateria de testes, foram acionadas todas as funcionalidades da aplicação, inclusive dentro dos menus, e realizadas sessões de jogo com todas as estruturas de entrada *default*, simulando uma rotina de uso habitual da aplicação. E, também, utilizando o aparelho conectado ao *desktop* por um cabo USB, foi realizada a adição de um novo arquivo de entrada e verificação da geração de arquivos de saída.

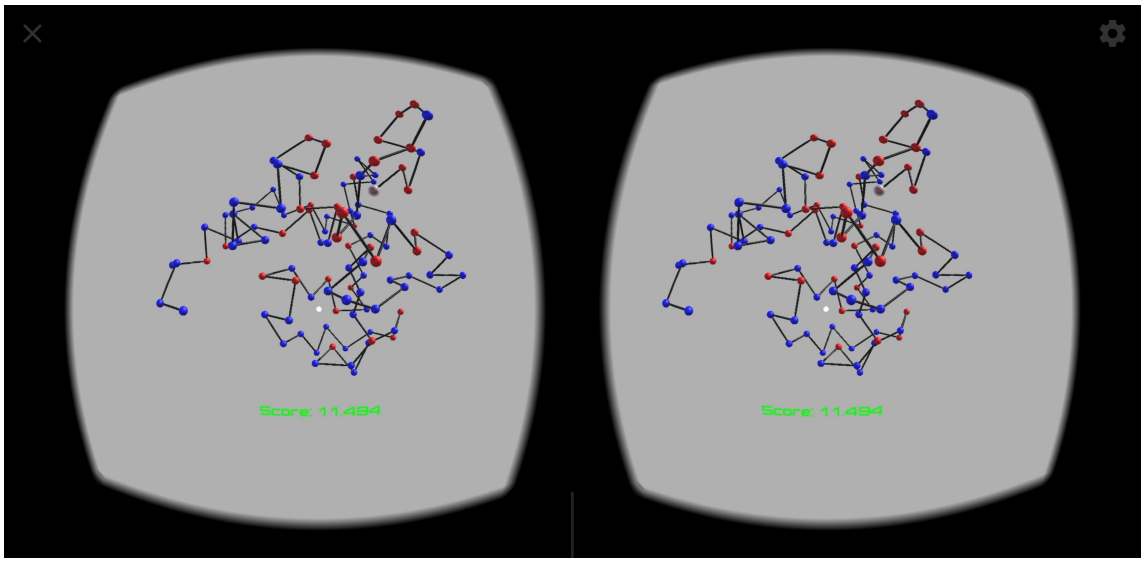
Desse modo, após a conclusão da bateria de testes, notou-se que, apesar da qualidade da imagem na tela do smartphone ser adequada, como pode ser visto na Figura 30, o equipamento de óculos VR não proporcionou uma imagem nítida e bem focalizada, causando desconforto visual ao usuário. Também foi possível notar que o consumo médio de bateria no

período de trinta minutos manteve-se entre oito e nove porcento, a cada execução. Sobre o aquecimento, esse foi pouco perceptível ao toque e o aparelho não indicou qualquer notificação de alerta. Por fim, não foram notados erros ou falhas do tipo atraso na imagem ou travamento da aplicação durante o procedimento.

Figura 30: Captura de imagem da tela do smartphone: (a) menu principal e (b) estrutura dentro do jogo.

(a)

(b)



Fonte: Autoria própria.

4.3 CONCLUSÕES ACERCA DOS RESULTADOS

Com a finalização do projeto, verifica-se com satisfação que todos os requisitos da especificação foram implementados e, também, a aplicação Protein Fold VR funciona sem erros ou falhas.

No mais, há algumas ressalvas que precisam ser pontuadas. A saber, a realização de testes em diferentes aparelhos (smartphones) configuraria uma maior garantia de compatibilidade da aplicação. Ainda, a qualidade da lente dos óculos causou desconforto considerável ao usuário, levando-se a considerar que investir em um equipamento de melhor qualidade seja a melhor opção, apesar do custo. E, mesmo com as pequenas variações observadas na estrutura, devido aos aspectos funcionais do motor de física em uma aplicação construída num ambiente de desenvolvimento de jogos, o resultado mostrou-se satisfatório, uma vez que as propriedades do modelo 3D-AB *off-lattice* foram mantidas nos arquivos de exportação.

Portanto, reitera-se a possibilidade de uso dos arquivos de saída por agentes interessados na pesquisa de dobramento de proteínas, bem como, utilizar a aplicação para fins didáticos, dada sua simplicidade e praticidade, inclusive por pessoas sem expertise na área da biologia computacional.

5 GESTÃO DO PROJETO

As seguintes seções apresentam os aspectos relacionados ao gerenciamento do tempo, orçamento e adversidades, adotado durante o desenvolvimento do projeto.

5.1 CRONOGRAMA

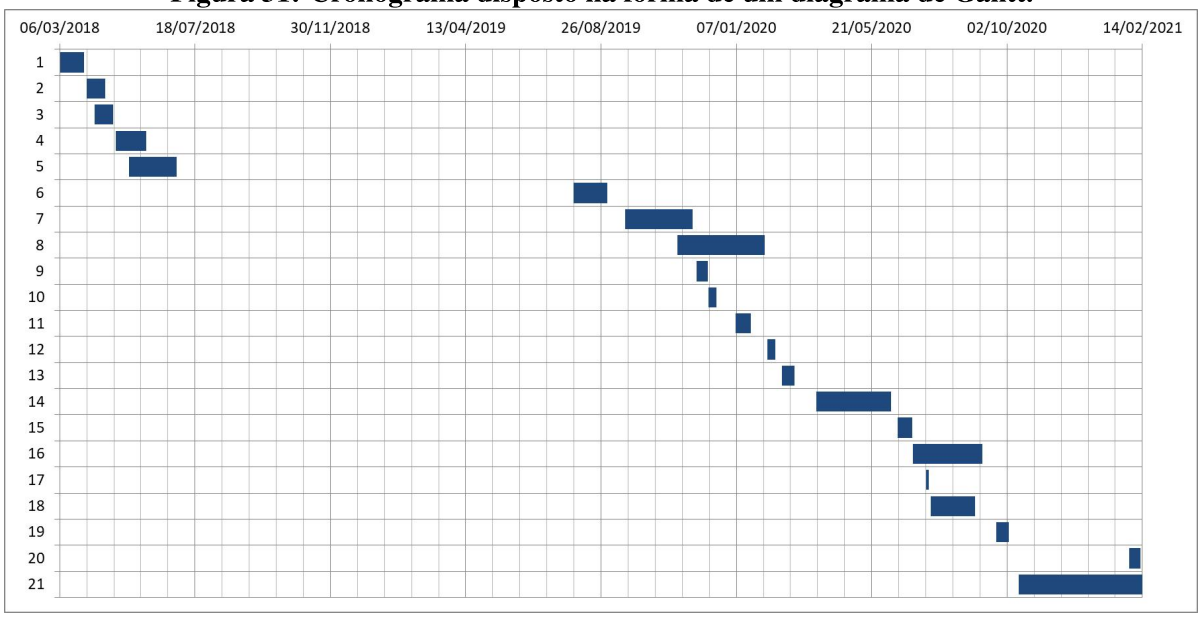
A Tabela 1 apresenta o cronograma de execução do projeto, que na Figura 31 também está esquematizado na forma de um diagrama de Gantt.

Tabela 1: Cronograma de execução do projeto.

ID	Ação	Início	Fim	Horas
1	Viabilidade e levantamento de requisitos	06/03/2018	30/03/2018	20
2	Estudo teórico sobre proteínas	02/04/2018	20/04/2018	12
3	Estudo do modelo 3D-AB *off-lattice*	10/04/2018	28/04/2018	12
4	Levantamento teórico sobre RV e gamificação	01/05/2018	31/05/2018	20
5	Estudo da plataforma Unity	14/05/2018	30/06/2018	40
6	Redação do referencial teórico	30/07/2019	01/09/2019	20
7	Implementação da movimentação e colisão dos objetos de jogo	19/09/2019	25/11/2019	25
8	Implementação dos *templates* de menus e do gerenciamento de arquivos	10/11/2019	05/02/2020	40
9	Implementação da rotina de seleção de resíduo	29/11/2019	10/12/2019	12
10	Implementação da movimentação da câmera	11/12/2019	19/12/2019	10
11	Implementação das rotinas de salvamento e carregamento	07/01/2020	22/01/2020	40
12	Implementação do *template* de janela modal	07/02/2020	15/02/2020	15
13	Implementação do teclado virtual para RV	22/02/2020	05/03/2020	20
14	Implementação da rotina de estabilização da estrutura	27/03/2020	09/06/2020	40
15	Implementação de rotina de verificação (para novos arquivos de entrada)	16/06/2020	30/06/2020	12
16	Implementação do menu de opções do *display*	01/07/2020	08/09/2020	20
17	Implementação do painel de informações da proteína	14/07/2020	17/07/2020	4
18	Implementação da função desfazer/refazer jogada	19/07/2020	01/09/2020	30
19	Implementação dos tutorias	22/09/2020	04/10/2020	10
20	Testes finais	01/02/2021	12/02/2021	8
21	Redação final do relatório	14/10/2020	14/02/2021	60
	Total	06/03/2018	14/02/2021	470

Fonte: Autoria própria

Figura 31: Cronograma disposto na forma de um diagrama de Gantt.



Fonte: Autoria própria.

5.2 CUSTOS

A Tabela 2 apresenta as despesas materiais envolvidas na execução do projeto.

Tabela 2: Relação dos custos materiais envolvidos no realização do projeto.

Material	Valor (R$)
Samsung Galaxy S8 Plus	2000
Kit VR BOX	40
Total	2040

Fonte: Autoria própria

5.3 RISCOS

Os principais riscos levantados para o projeto estão descritos abaixo:

Risco 1

Importância: Alta.

Descrição: Excesso de trabalho para um aluno.

Efeito no Projeto: Atraso no desenvolvimento.

Probabilidade: Média.

Impacto: Médio.
Ação: Mitigar. Definição clara de cronograma.

Risco 2
Importância: Alta.
Descrição: Falha na integração entre o software a ser desenvolvido e o RV.
Efeito no Projeto: Ferramenta de interface com o usuário terá de ser modificada.
Probabilidade: Baixa.
Impacto: Alto.
Ação: Eliminar. Estudo aprofundado de técnicas de utilização do óculos RV e de criação de software para Android.

Risco 3
Importância: Baixa.
Descrição: Falta de capacidade de processamento em um dispositivo smartphone para gerar imagens para RV em 60 fps.
Efeito no Projeto: O número de fps no RV terá de ser diminuído, podendo criar desconforto no usuário.
Probabilidade: Baixa.
Impacto: Baixo.
Ação: Eliminar. Uso de um smartphone com melhor capacidade de processamento.

Risco 4
Importância: Alta.
Descrição: Problemas no dispositivo smartphone.
Efeito no Projeto: Atraso no desenvolvimento devido a ter de comprar novo dispositivo.
Probabilidade: Baixa.
Impacto: Alto.
Ação: Mitigar. Criar o software de forma que seja compatível com o maior número de smartphones possível.

Risco 5
Importância: Alta.
Descrição: Problemas nos dispositivos que compõem os óculos RV.

Efeito no Projeto: Atraso no desenvolvimento devido a ter de comprar novos dispositivos.
Probabilidade: Baixa.
Impacto: Alto.
Ação: Conviver. Procurar formas diferentes de realizar os testes enquanto os novos dispositivos não estejam em mãos.

Risco 6
Importância: Média.
Descrição: Falha na comunicação *bluetooth* entre o *joystick* e o celular.
Efeito no Projeto: Falta de um dispositivo de interação com o usuário.
Probabilidade: Baixa.
Impacto: Médio.
Ação: Eliminar. Ligar o dispositivo de interação com o usuário através de internet *wireless* ou cabo.

Risco 7
Importância: Alta.
Descrição: Falha na gamificação da estrutura proteica utilizando a plataforma Unity.
Efeito no Projeto: O usuário não poderá alterar a estrutura proteica.
Probabilidade: Baixa.
Impacto: Alto.
Ação: Eliminar. Buscar outra plataforma que possa ser utilizada para a gamificação da estrutura proteica ou implementar as interações sem o auxílio de uma plataforma.

6 CONSIDERAÇÕES FINAIS

Apesar das dificuldades encontradas durante o entretempo, considera-se a aplicação Protein Fold VR e sua experiência de desenvolvimento um fechamento muito satisfatório para o curso de Engenharia de Computação. Por meio dos testes, foi possível melhorar a energia interna em todas as estruturas, principalmente naquelas com mais aminoácidos, indicando que essa é uma forma válida e, ao mesmo tempo, interessante de abordar o Problema do Dobramento de Proteínas.

Além do mais, deseja-se apontar aspectos que podem ser aperfeiçoados em trabalhos futuros. O primeiro deles é a precisão. No projeto foi utilizado o tipo *float* para realizar os cálculos de energia e armazenar as posições dos resíduos, isso porque a Unity possui muitas funções nativas otimizadas para esse tipo, como cálculos de módulo, normalização e produto interno e externo de vetores. No caso de ser utilizado o tipo *double*, que é suportado pela plataforma, para aumentar a precisão nos cálculos, teriam de ser revistas as funções de cálculo de pontuação e centro de massa, por exemplo. E levando em conta que a Unity não possui tantas funcionalidades nativas para esse tipo, seria necessário implementar funções próprias e avaliar o impacto no desempenho.

Ainda sobre aperfeiçoamentos futuros, poderia ser realizado um investimento para que o aplicativo fosse disponibilizado na loja Google Play. No entanto, seria necessário fazer uma revisão para formatar a aplicação de acordo com as políticas da loja e pagar a taxa de registro para desenvolvedores.

Sobre a interação com o usuário, um ponto a ser explorado é a implementação de funcionalidades fora do ambiente de RV, como a inclusão de uma nova estrutura, a exportação das estruturas de saída e um modo em que o usuário interage por meio de *touchscreen* com as funcionalidades da aplicação. Também, pode-se incluir outros elementos de gamificação, como uma contagem regressiva de tempo para alcançar certa pontuação ou corrigir uma pontuação negativa. No que tange à interface, há a possibilidade de implementar a funcionalidade de mudança de idioma e também seleção de coloração de fundo, bem como o mapeamento dos

botões do *joystick* pelo usuário. Por fim, há a possibilidade de conectar a aplicação via internet para baixar novas estruturas de um servidor próprio da aplicação.

Em conclusão, reitera-se a capacidade da ferramenta para visualizar e manipular estruturas simplificadas de proteínas, em um ambiente de realidade virtual no qual a interação com o usuário é caracterizada pela simplicidade e intuitividade dos jogos eletrônicos. E, assim, encerra-se a proposta do trabalho, deixando-se seu código disponibilizado na plataforma GitHub (`https://github.com/lucasdf09/TCC_PROTEIN_FOLD_VR`), para eventuais interessados, tanto para fins acadêmicos, quanto pedagógicos ou meramente recreativos.

REFERÊNCIAS

AGUZZI, A.; O'CONNOR, T. Protein aggregation diseases: Pathogenicity and therapeutic perspectives. **Nature Reviews Drug Discovery**, 2010.

ALBERTS, B. et al. **Biologia molecular da célula**. 6. ed. Porto Alegre: Artmed, 2017.

ANFINSEN, C. B. Principles that govern the folding of protein chains. **Science**, v. 181, n. 4096, p. 223–230, 1973.

ANFINSEN, C. B. et al. The kinetics of formation of native ribonuclease during oxidation of the reduced polypeptide chain. **Proceedings of the National Academy of Sciences of the United States of America**, v. 47, n. 9, p. 1309–1314, 1961.

ARAUJO, R. B. d.; KIRNER, C. **Especificação e análise de um sistema distribuído de realidade virtual**. Dissertação (Mestrado) — Universidade de São Paulo, 1996.

BENÍTEZ, C.; LOPES, H. Molecular dynamics for simulating the protein folding process using the 3D-AB off-lattice model. In: **Advances in Bioinformatics and Computational Biology**. Heidelberg: Springer, 2012. p. 61–72.

BENÍTEZ, C.; LOPES, H. Ab-initio protein folding using molecular dynamics and a simplified off-lattice model. **Journal of Bionanoscience**, v. 7, p. 391–402, 2013.

BENÍTEZ, C. M. V. **Contributions to the study of the protein folding problem using bioinspired computation and molecular dynamics**. 193 p. Tese (Doutorado) — Federal University Of Technology - Paraná, 2015.

BRYSON, S. Virtual reality in scientific visualization. **Commun. ACM**, Association for Computing Machinery, New York, NY, USA, v. 39, n. 5, p. 62–71, maio 1996.

CAILLOIS, R.; BARASH, M. **Man, Play, and Games**. Urbana, Chicago: University of Illinois Press, 2001. (Sociology/Sport).

CONTESSOTO, V. de G. et al. Introdução ao problema de enovelamento de proteínas: uma abordagem utilizando modelos computacionais simplificados. **Revista Brasileira de Ensino de Física**, v. 40, n. 4, 2018.

COOPER, G. M.; HAUSMAN, R. E. **The cell: a molecular approach**. 4. ed. Washington D.C.: ASM Press, 2007.

DETERDING, S. et al. From game design elements to gamefulness: Defining "gamification". In: **Proceedings of the 15th International Academic MindTrek Conference: Envisioning Future Media Environments**. New York, NY, USA: Association for Computing Machinery, 2011. (MindTrek '11), p. 9–15.

DETERDING, S. et al. Gamification: Toward a definition. **Proceedings of CHI 2011 Workshop Gamification: Using Game Design Elements in Non-Game Contexts**, p. 6–9, 01 2011.

DILL, K. A.; MACCALLUM, J. L. The protein-folding problem, 50 years on. **Science**, American Association for the Advancement of Science, v. 338, n. 6110, p. 1042–6, nov 2012.

DILL, K. A. et al. The protein folding problem. **Annual Review of Biophysics**, 2008.

FRASER-PITT, D.; O'NEIL, D. Cystic fibrosis - A multiorgan protein misfolding disease. **Future Science OA**, v. 1, n. 2, 2015.

GoogleVR. **GvrReticlePointer prefab**. 2019. Acessado em 10/05/2019. Disponível em: <https://developers.google.com/vr/reference/unity/prefab/GvrReticlePointer>.

GRIFFITHS, A. J. F. et al. **Introdução à genética**. 11. ed. Rio de Janeiro: Guanabara Koogan, 2016.

GROH, F. Gamification: State of the art definition and utilization. **Proceedings of the 4th Seminar on Research Trends in Media Informatics**, p. 39–46, 01 2012.

HALLIDAY, D.; RESNICK, R.; WALKER, J. **Fundamentos de física, volume 1 : mecânica**. 9. ed. Rio de Janeiro: LTC, 2012.

HATTORI, L. T.; BENÍTEZ, C.; LOPES, H. A deep bidirectional long short-term memory approach applied to the protein secondary structure prediction problem. In: **2017 IEEE Latin American Conference on Computational Intelligence (LA-CCI)**. Arequipa: [s.n.], 2017. p. 1–6.

HUIZINGA, J. **Homo ludens: o jôgo como elemento da cultura**. São Paulo: Editora da Universidade de S. Paulo, Editora Perspectiva, 1971. (Coleção estudos).

IMARISIO, S. et al. Huntington's disease: From pathology and genetics to potential therapies. **Biochemical Journal**, v. 412, n. 2, p. 191–209, 2008.

IRBÄCK, A. et al. Local interactions and protein folding: A three-dimensional off-lattice approach. **Journal of Chemical Physics**, v. 107, n. 1, p. 273–282, 1997.

KHATIB, F. et al. Crystal structure of a monomeric retroviral protease solved by protein folding game players. **Nature structural & molecular biology**, v. 18, p. 1175–7, 09 2011.

KIRNER, C.; SISCOUTTO, R. Realidade virtual e aumentada: conceitos, projeto e aplicações. In: **Livro do IX Symposium on Virtual and Augmented Reality, Petrópolis (RJ)**. Porto Alegre: SBC, 2007. v. 28.

LEOPOLD, P. E.; MONTAL, M.; ONUCHIC, J. N. Protein folding funnels: A kinetic approach to the sequence-structure relationship. **Proceedings of the National Academy of Sciences of the United States of America**, v. 89, p. 8721–8725, 1992.

LEVINTHAL, C. Are there pathways for protein folding? **Journal de Chimie Physique**, 1968.

LIU, Y.; ALEXANDROVA, T.; NAKAJIMA, T. Gamifying intelligent environments. In: **Proceedings of the 2011 International ACM Workshop on Ubiquitous Meta User Interfaces**. New York, NY, USA: Association for Computing Machinery, 2011. (Ubi-MUI '11), p. 7–12.

LODISH, H. et al. **Molecular cell biology**. 8th ed.. ed. New York: W. H. Freeman and Company, 2016.

MACHOVER, C.; TICE, S. E. Virtual reality. **IEEE Computer Graphics and Applications**, v. 14, n. 1, p. 15–16, 1994.

MALONE, T. W. Toward a theory of intrinsically motivating instruction. **Cognitive Science**, v. 5, n. 4, p. 333–369, 1981.

MASCIONI, A. et al. Conformational preferences of the amylin nucleation site in SDS micelles: an NMR study. **Biopolymers**, v. 69, p. 29–41, 2003.

MCGONIGAL, J. **Jane McGonigal: Gaming can make a better world — TED Talk**. 2010. Acessado em 17/05/2018. Disponível em: <https://www.ted.com/talks/jane_mcgonigal_gaming_can_make_a_better_world?language=en>.

MCGONIGAL, J. **Reality Is Broken: Why Games Make Us Better and How They Can Change the World**. New York: Penguin Publishing Group, 2011.

Microsoft. **O que é Realidade Misturada? - Mixed Reality — Microsoft Docs**. 2020. Acessado em 06/02/2020. Disponível em: <https://docs.microsoft.com/pt-br/windows/mixed-reality/discover/mixed-reality>.

MILGRAM, P.; KISHINO, F. A taxonomy of mixed reality visual displays. **IEICE Trans. Information Systems**, vol. E77-D, no. 12, p. 1321–1329, 12 1994.

NELSON, D. L.; COX, M. M. **Princípios de bioquímica de Lehninger**. 6. ed. Porto Alegre: Artmed, 2014.

NETTO, A. V.; MACHADO, L. d. S.; OLIVEIRA, M. C. F. d. Realidade virtual - definições, dispositivos e aplicações. **REIC - Revista Eletrônica de Iniciação Científica**, 2002.

NGO, J. T.; MARKS, J.; KARPLUS, M. Computational complexity, protein structure prediction, and the levinthal paradox. In: **The Protein Folding Problem and Tertiary Structure Prediction**. Boston, MA: Birkhäuser Boston, 1994. p. 433–506.

PIMENTEL, K.; TEIXEIRA, K. **Virtual Reality: Through the New Looking Glass**. USA: McGraw-Hill, Inc., 1993.

Prodview. **Estudo revela que o mercado de games alcança 70% da população digital brasileira - Prodview**. 2020. Acessado em 02/02/2020. Disponível em: <https://prodview.com.br/2020/07/24/estudo-revela-que-o-mercado-de-games-alcanca-70-da-populacao-digital-brasileira/>.

RAMASWAMY, S.; SHANNON, K. M.; KORDOWER, J. H. Huntington's disease: Pathological mechanisms and therapeutic strategies. **Cell Transplantation**, v. 16, n. 3, p. 301–312, 2007.

RAPAPORT, D. C. et al. The Art of Molecular Dynamics Simulation. **Computers in Physics**, 1996.

SALEN, K.; ZIMMERMAN, E. **Rules of Play: Game Design Fundamentals**. USA: The MIT Press, 2003.

STILLINGER, F. H.; HEAD-GORDON, T. Collective aspects of protein folding illustrated by a toy model. **Physical Review E**, v. 52, n. 3, p. 2872–2877, 1995.

Unity. **Unity - Manual: Asset Workflow**. 2018. Acessado em 09/05/2019. Disponível em: <https://docs.unity3d.com/2019.2/Documentation/Manual/AssetWorkflow.html>.

Unity. **Input Field — Unity UI — 1.0.0**. 2019. Acessado em 21/09/2019. Disponível em: <https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/script-InputField.html>.

Unity. **Toggle — Unity UI — 1.0.0**. 2019. Acessado em 21/09/2019. Disponível em: <https://docs.unity3d.com/Packages/com.unity.ugui@1.0/manual/script-Toggle.html>.

Unity. **Unity - Manual: Coroutines**. 2019. Acessado em 21/09/2019. Disponível em: <https://docs.unity3d.com/2019.2/Documentation/Manual/Coroutines.html>.

Unity. **Unity - Manual: Creating Prefabs**. 2019. Acessado em 05/05/2019. Disponível em: <https://docs.unity3d.com/2019.2/Documentation/Manual/CreatingPrefabs.html>.

Unity. **Unity - Manual: Introduction to components**. 2019. Acessado em 05/05/2019. Disponível em: <https://docs.unity3d.com/2019.2/Documentation/Manual/Components.html>.

Unity. **Unity - Manual: Joints**. 2019. Acessado em 09/05/2019. Disponível em: <https://docs.unity3d.com/2019.2/Documentation/Manual/Joints.html>.

Unity. **Unity - Manual: Rigidbody**. 2019. Acessado em 09/05/2019. Disponível em: <https://docs.unity3d.com/2019.2/Documentation/Manual/class-Rigidbody.html>.

Unity. **Unity - Manual: Scenes**. 2019. Acessado em 12/06/2019. Disponível em: <https://docs.unity3d.com/2019.2/Documentation/Manual/CreatingScenes.html>.

Unity. **Unity - Manual: Streaming Assets**. 2019. Acessado em 11/05/2019. Disponível em: <https://docs.unity3d.com/2019.2/Documentation/Manual/StreamingAssets.html>.

Unity. **Unity - Scripting API: Input**. 2019. Acessado em 21/09/2019. Disponível em: <https://docs.unity3d.com/2019.2/Documentation/ScriptReference/Input.html>.

Unity. **Unity - Scripting API: Input.GetJoystickNames**. 2019. Acessado em 21/09/2019. Disponível em: <https://docs.unity3d.com/2019.2/Documentation/ScriptReference/Input.GetJoystickNames.html>.

Unity. **Unity - Scripting API: MonoBehaviour.FixedUpdate()**. 2019. Acessado em 21/09/2019. Disponível em: <https://docs.unity3d.com/2019.2/Documentation/ScriptReference/MonoBehaviour.FixedUpdate.html>.

Unity. **Unity - Scripting API: Physics.autoSimulation**. 2019. Acessado em 21/09/2019. Disponível em: <https://docs.unity3d.com/2019.2/Documentation/ScriptReference/Physics-autoSimulation.html>.

Unity. **Unity - Scripting API: PlayerPrefs**. 2019. Acessado em 12/06/2019. Disponível em: <https://docs.unity3d.com/2019.2/Documentation/ScriptReference/PlayerPrefs.html>.

Unity. **Unity - Scripting API: Rigidbody.isKinematic**. 2019. Acessado em 21/09/2019. Disponível em: <https://docs.unity3d.com/2019.2/Documentation/ScriptReference/Rigidbody-isKinematic.html>.

Unity. **Unity - Scripting API: Transform.LookAt**. 2019. Acessado em 21/09/2019. Disponível em: <https://docs.unity3d.com/2019.2/Documentation/ScriptReference/Transform.LookAt.html>.

Unity. **Unity - Scripting API: Transform.RotateAround**. 2019. Acessado em 21/09/2019. Disponível em: <https://docs.unity3d.com/2019.2/Documentation/ScriptReference/Transform.RotateAround.html>.

Unity. **Unity - Scripting API: Transform.Translate**. 2019. Acessado em 21/09/2019. Disponível em: <https://docs.unity3d.com/2019.2/Documentation/ScriptReference/Transform.Translate.html>.

Unity. **Unity - Scripting API: XR.InputTracking.Recenter**. 2019. Acessado em 21/09/2019. Disponível em: <https://docs.unity3d.com/2019.2/Documentation/ScriptReference/XR.InputTracking.Recenter.html>.

Unity. **Unity - Manual: Google VR hardware and software requirements**. 2020. Acessado em 08/07/2020. Disponível em: <https://docs.unity3d.com/2019.2/Documentation/Manual/googlevr_requirements.html>.

Wareable. **Vomit Reality: Why VR makes some of us feel sick and how to make it stop**. 2016. Acessado em 06/02/2020. Disponível em: <https://www.wareable.com/vr/vr-headset-motion-sickness-solution-777>.

Wired. **The Untold Story of Magic Leap, the World's Most Secretive Startup — WIRED**. 2016. Acessado em 06/02/2020. Disponível em: <https://www.wired.com/2016/04/magic-leap-vr/>.

ZHENG, J. M.; CHAN, K. W.; GIBSON, I. Virtual reality. **IEEE Potentials**, v. 17, n. 2, p. 20–23, 1998.

ZICHERMANN, G.; CUNNINGHAM, C. **Gamification by Design: Implementing Game Mechanics in Web and Mobile Apps**. 1. ed. Sebastopol, CA: O'Reilly Media, Inc., 2011.

ZWANZIG, R.; SZABO, A.; BAGCHI, B. Levinthal's paradox. **Proceedings of the National Academy of Sciences**, National Academy of Sciences, v. 89, n. 1, p. 20–22, 1992.

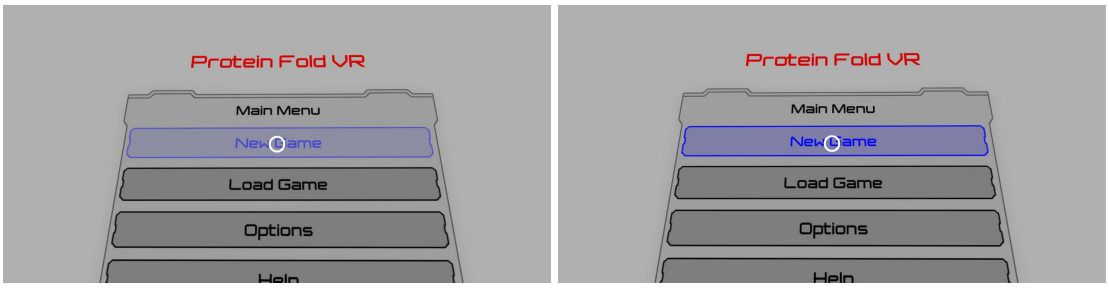
ANEXO A – ARQUIVO DE ENTRADA 10_1KUW

```
0  500001.603654  500001.717158  500002.459935
1  500002.011537  500000.877928  500002.100318
2  500001.570893  500001.451715  500001.409956
3  500001.643580  500000.509671  500001.082438
4  500001.062047  500001.085131  500000.507403
5  500001.372027  500000.346745  499999.908488
6  500000.794739  500001.023441  499999.451520
7  500000.839065  500000.160343  499998.948433
8  500000.226285  500000.821018  499998.514831
9  500000.361603  499999.965051  499998.015819
BestEnergy = -11.638971
BestBondEnergy = -3.012316
BestTorsionEnergy = -3.385401
BestLJEnergy = -5.241254
BestInterEnergy = 0.000000
Step = 4700270
rGAll = 1.636609
rGH = 0.924884
rGP = 1.905785
rGAll_inter = 0.000000
rGH_inter = 0.000000
rGP_inter = 0.000000
Processing Time = 1466.000000s
Molecules = 10
Sequence = BBBABAAABB
```

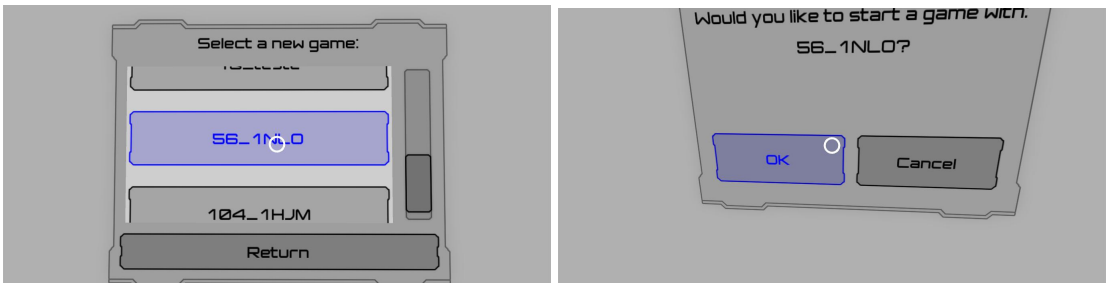
ANEXO B – EXEMPLO DE *GAMEPLAY* (*SCREENSHOTS*)

Figura 32: (a) Seleção de novo jogo no Menu Principal. (b) Clique no botão de novo jogo. (c) Escolha da estrutura. (d) Confirmação.

(a) (b)



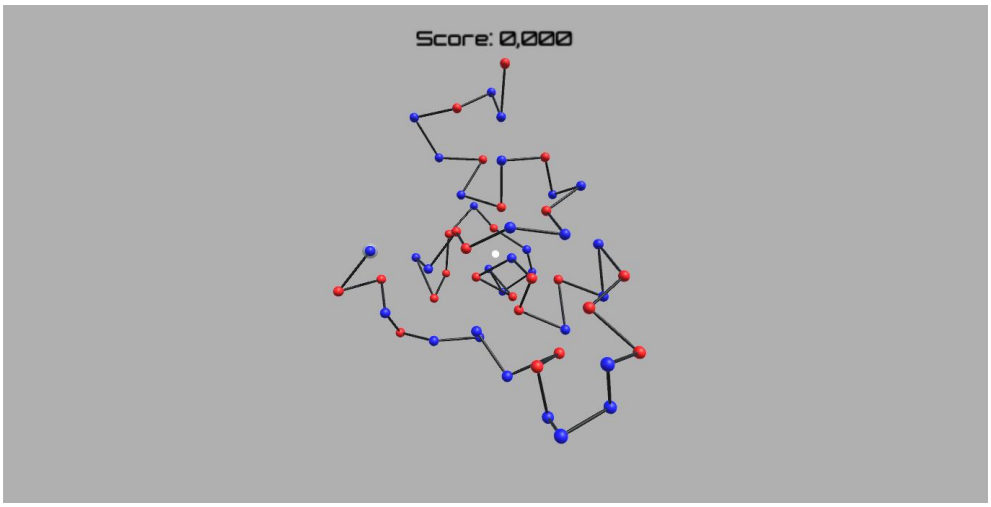
(c) (d)



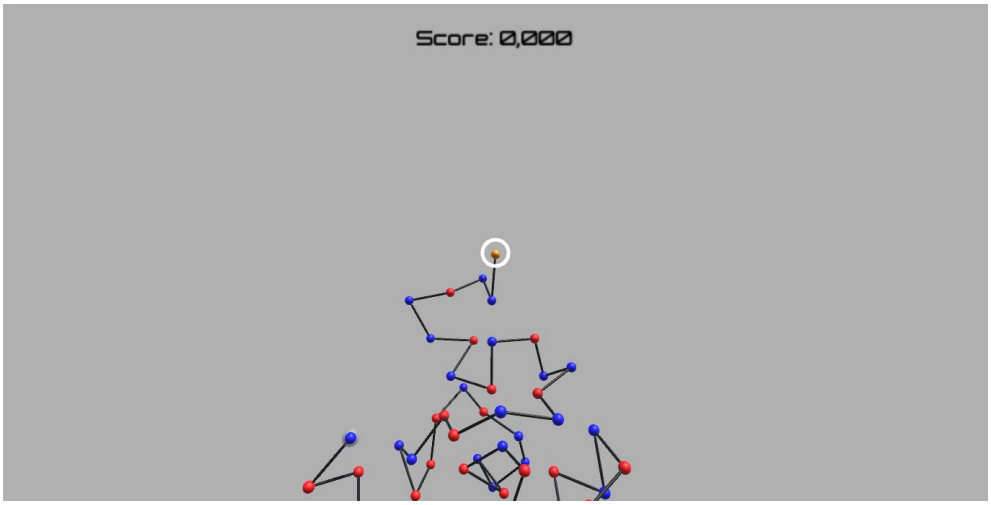
Fonte: Autoria própria.

Figura 32: (e) Ambiente de jogo, modo de seleção. (f) Posicionamento do *pointer* em resíduo a ser movimentado. (g) Resíduo selecionado, modo de movimento.

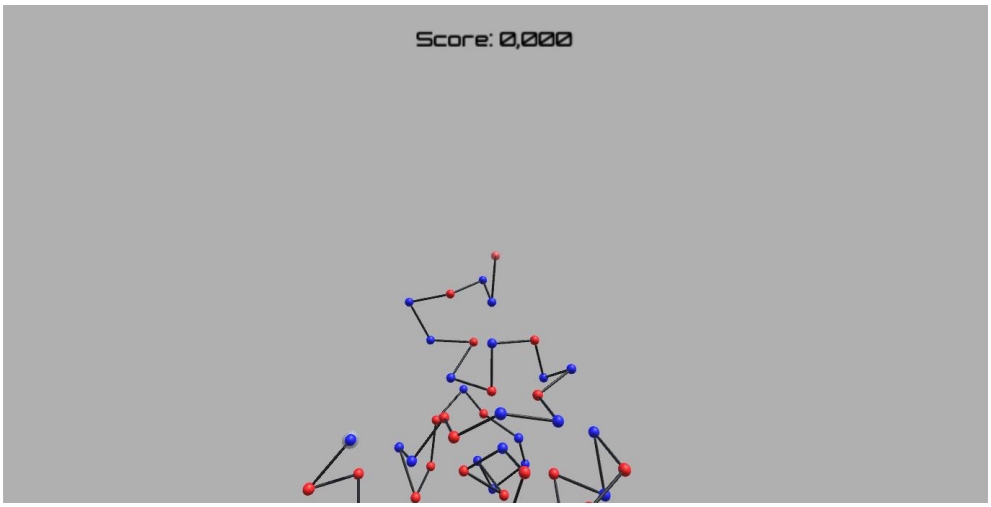
(e)



(f)



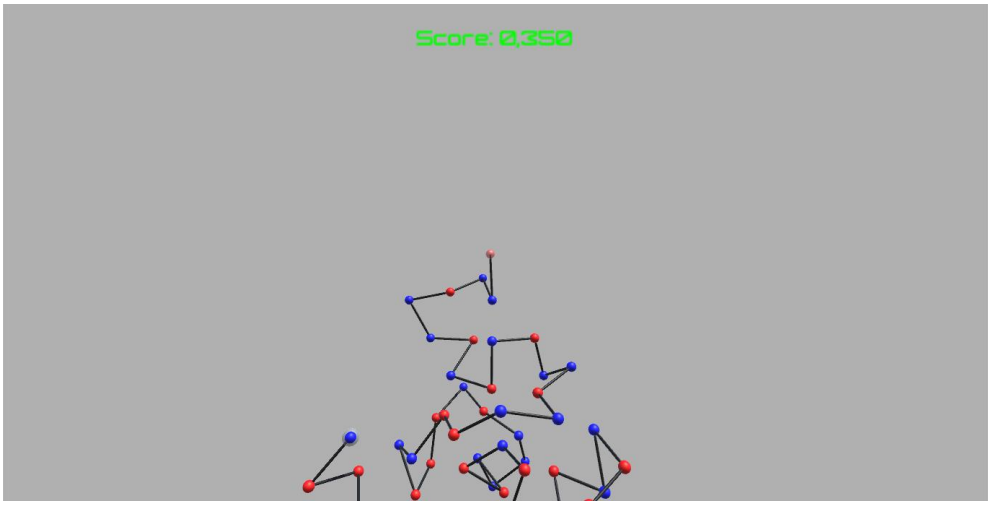
(g)



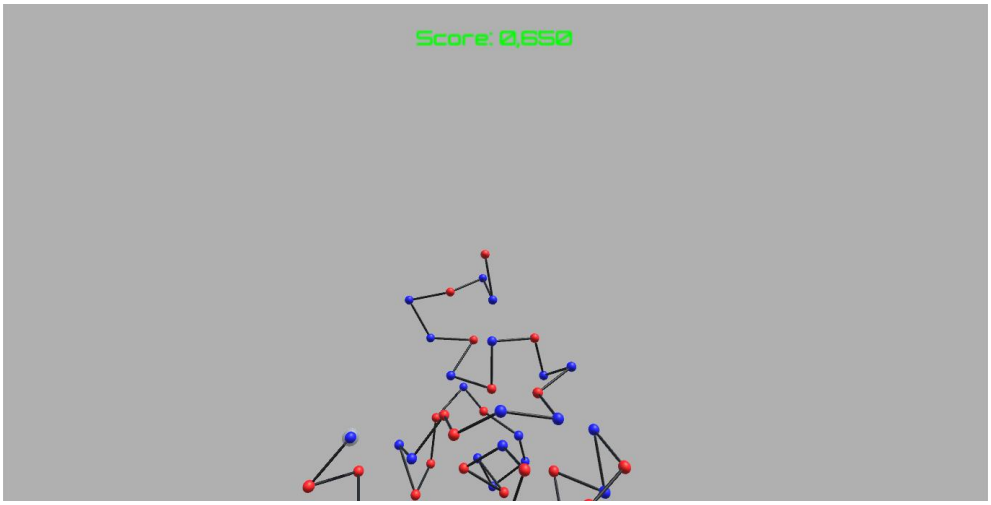
Fonte: Autoria própria.

Figura 32: (h), (i) e (j) Movimentação no sentido do direcional para a esquerda (←).

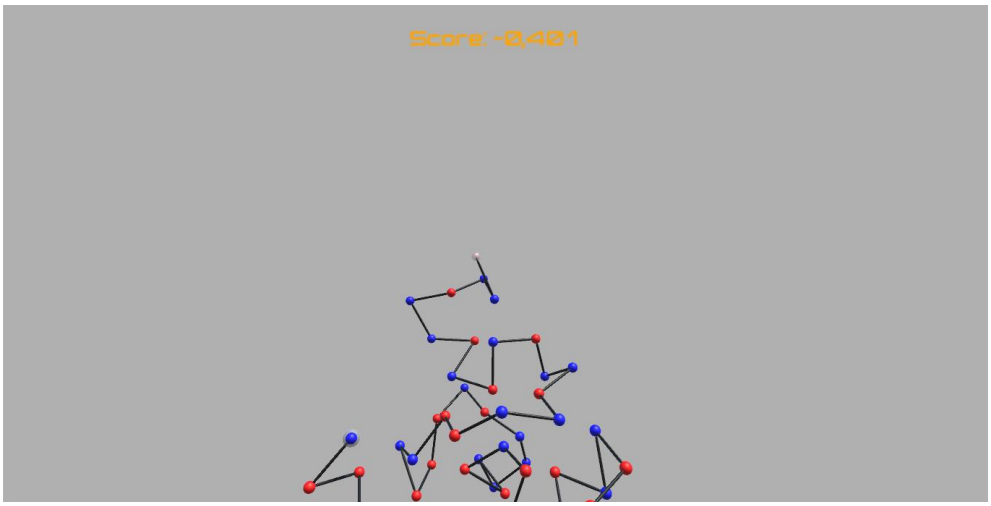
(h)



(i)



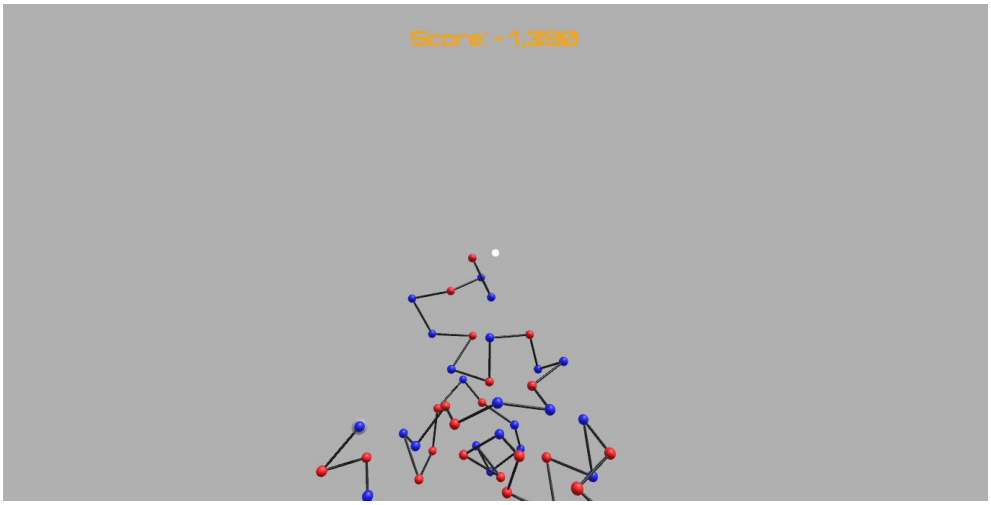
(j)



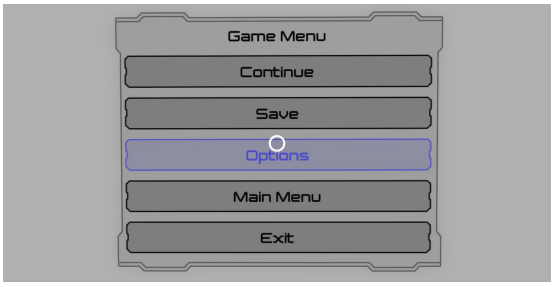
Fonte: Autoria própria.

Figura 32: (k) Resíduo desmarcado, modo de seleção. (l) Menu de Jogo. (m) Sair. (n) Confirmação.

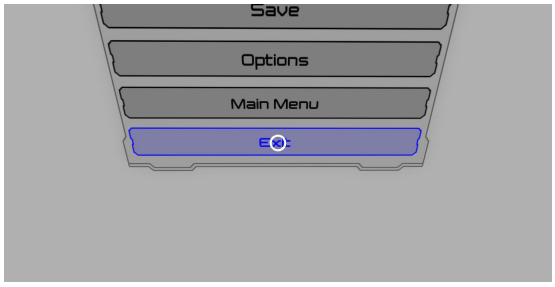
(k)



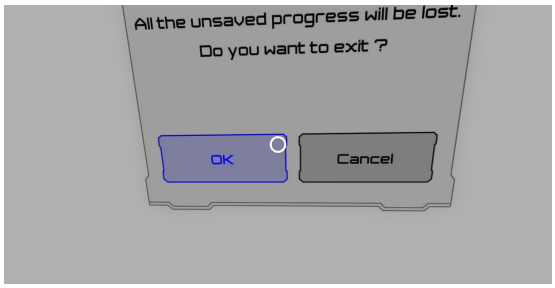
(l)



(m)



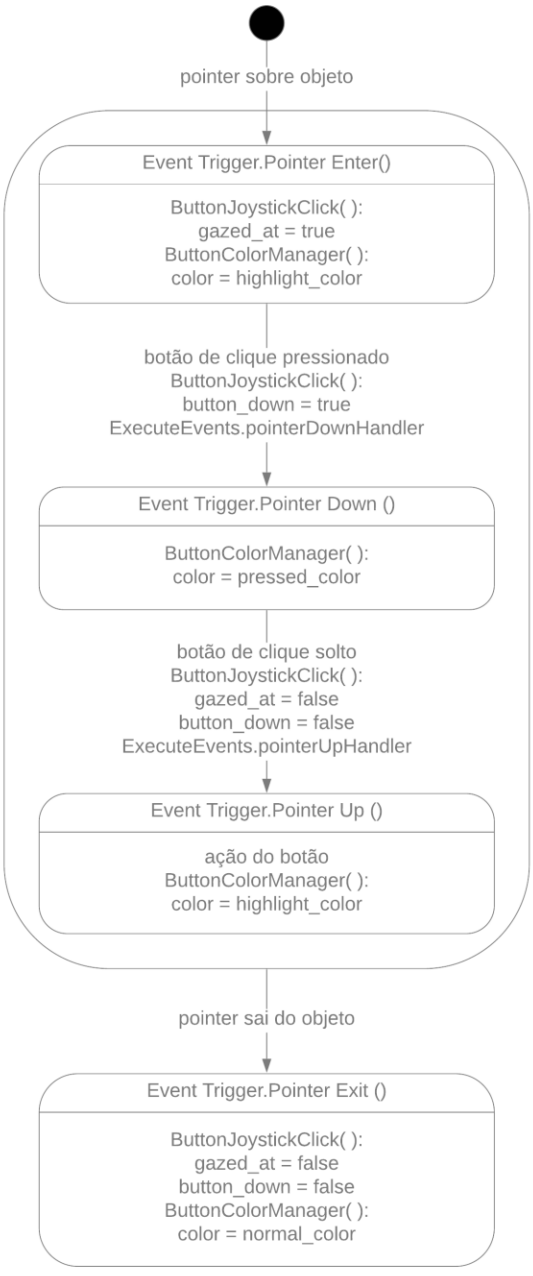
(n)



Fonte: Autoria própria.

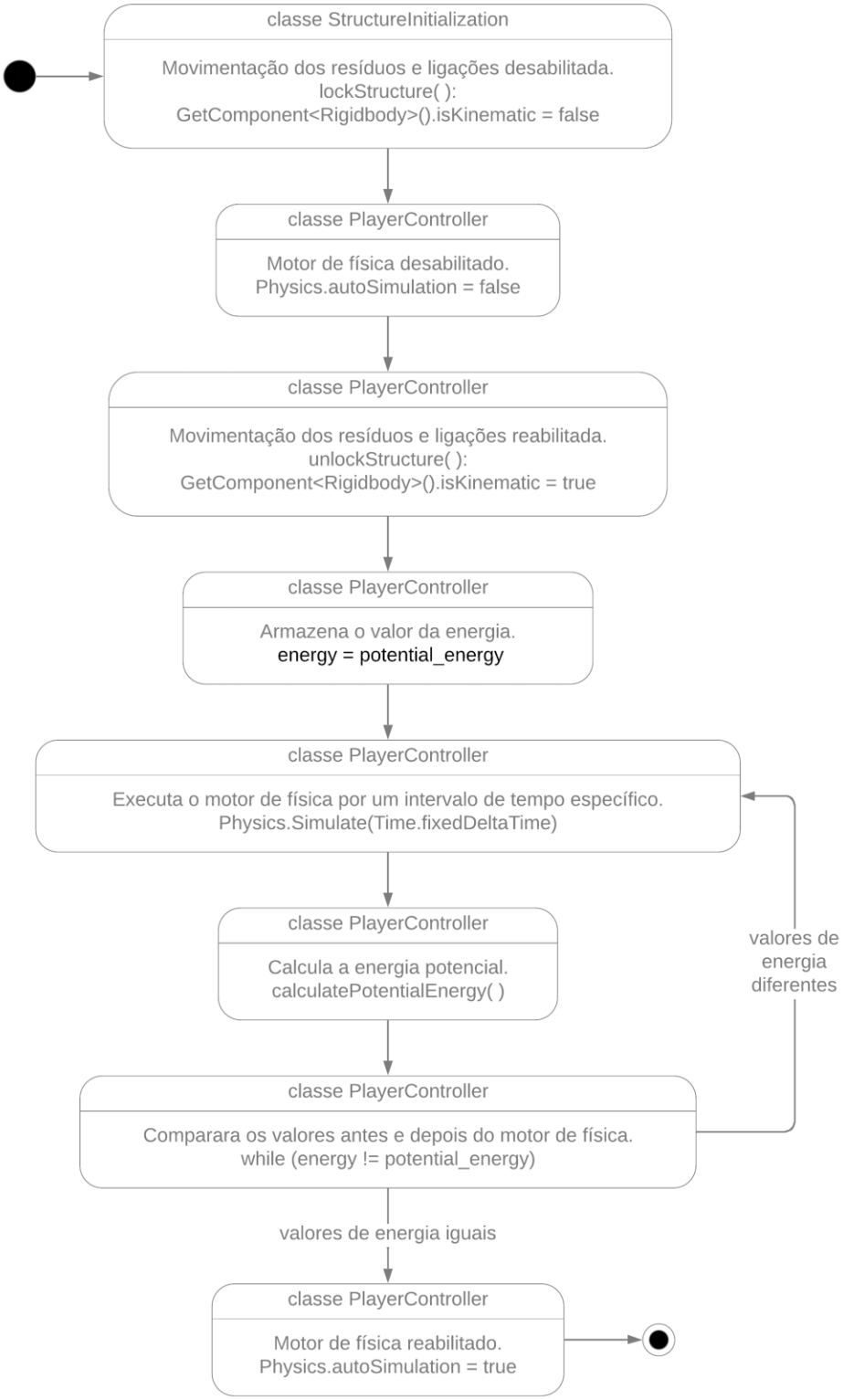
ANEXO C – DIAGRAMAS DE ATIVIDADE

Figura 33: Diagrama de evento de clique.



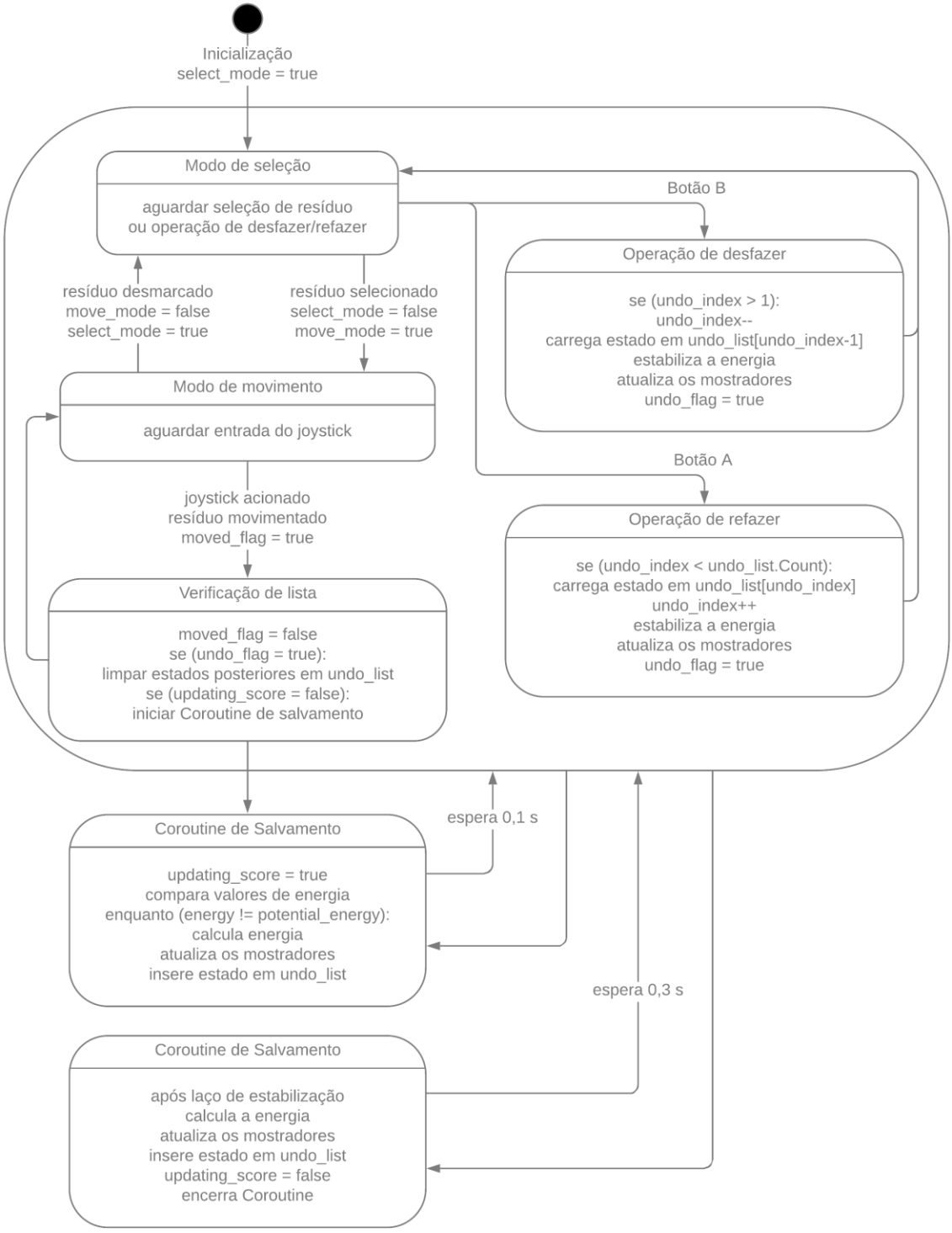
Fonte: Autoria própria.

Figura 34: Diagrama de inicialização.



Fonte: Autoria própria.

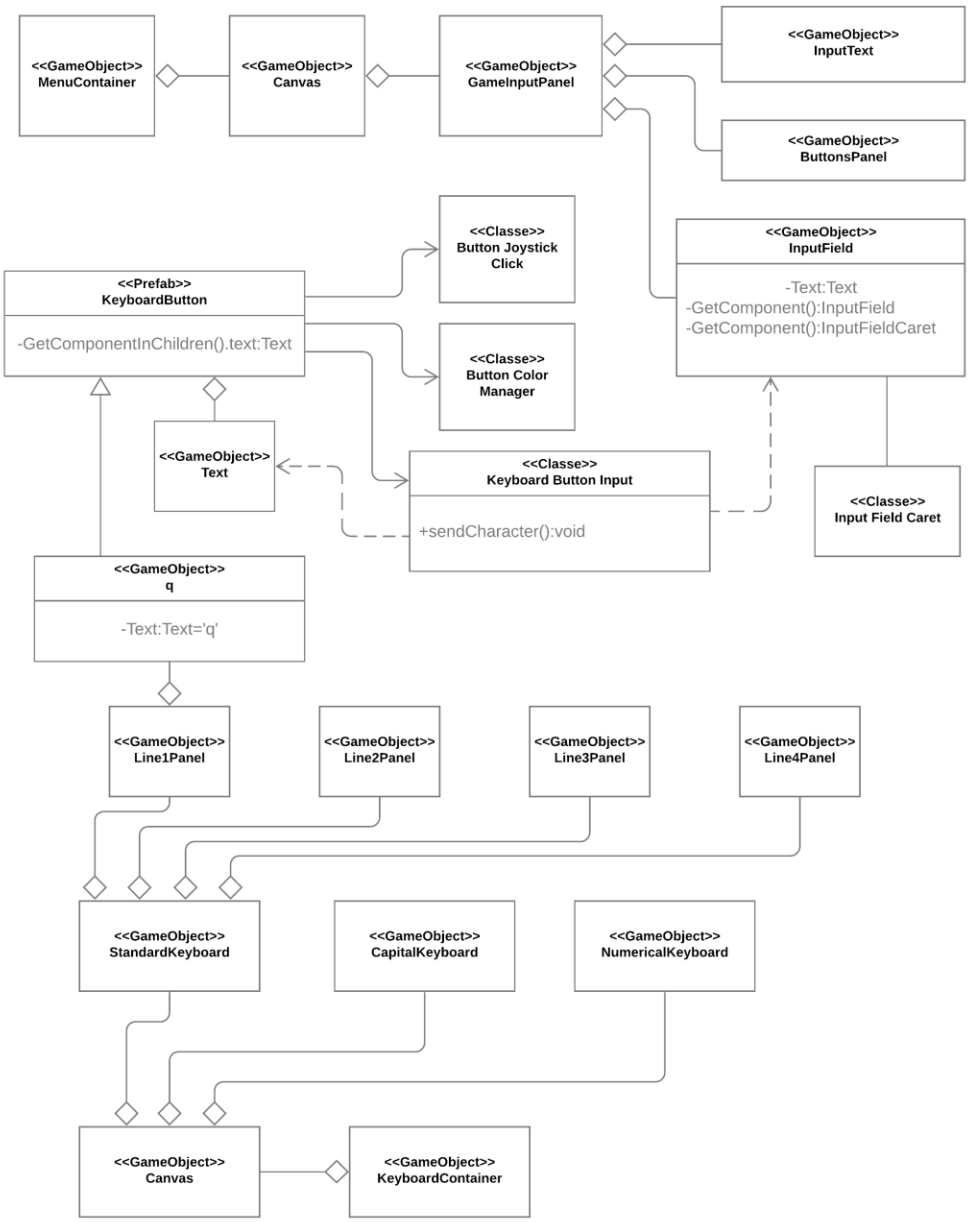
Figura 35: Diagrama da função desfazer/refazer.



Fonte: Autoria própria.

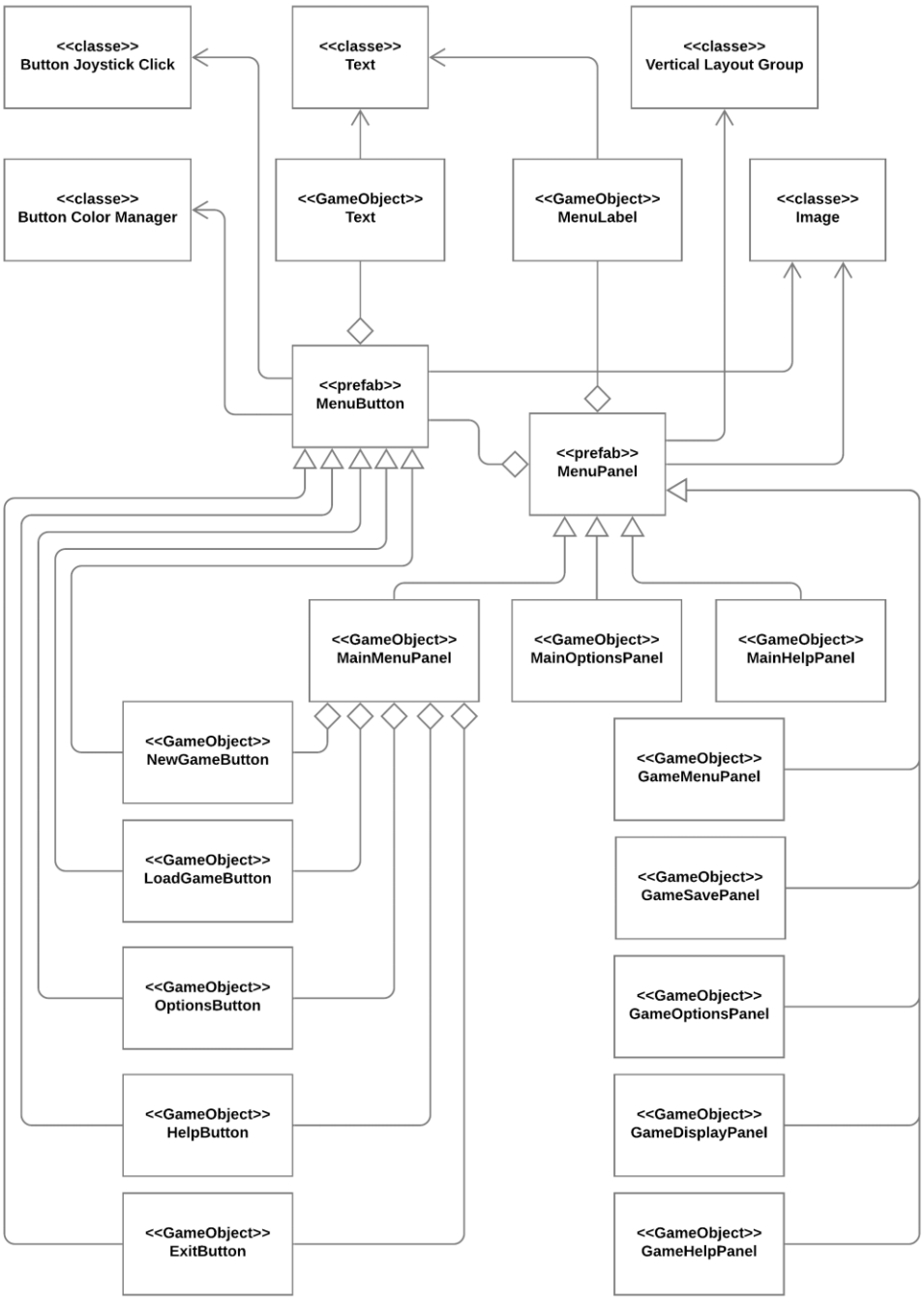
ANEXO D – DIAGRAMAS DE CLASSE E OBJETOS ADAPTADOS PARA O SISTEMA ENTIDADE COMPONENTE DA UNITY

Figura 36: Diagrama do teclado virtual.



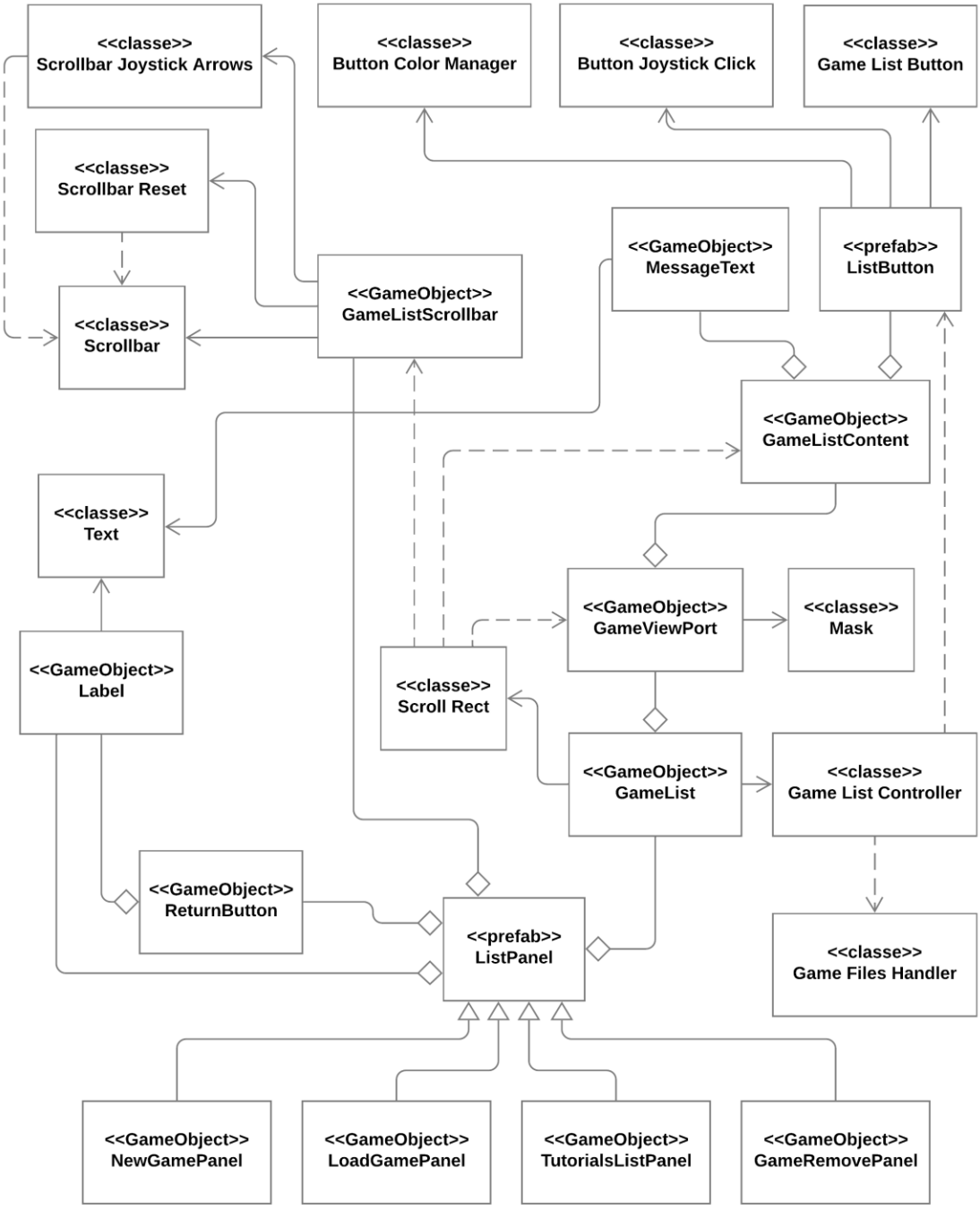
Fonte: Autoria própria.

Figura 37: Diagrama do prefab MenuPanel.



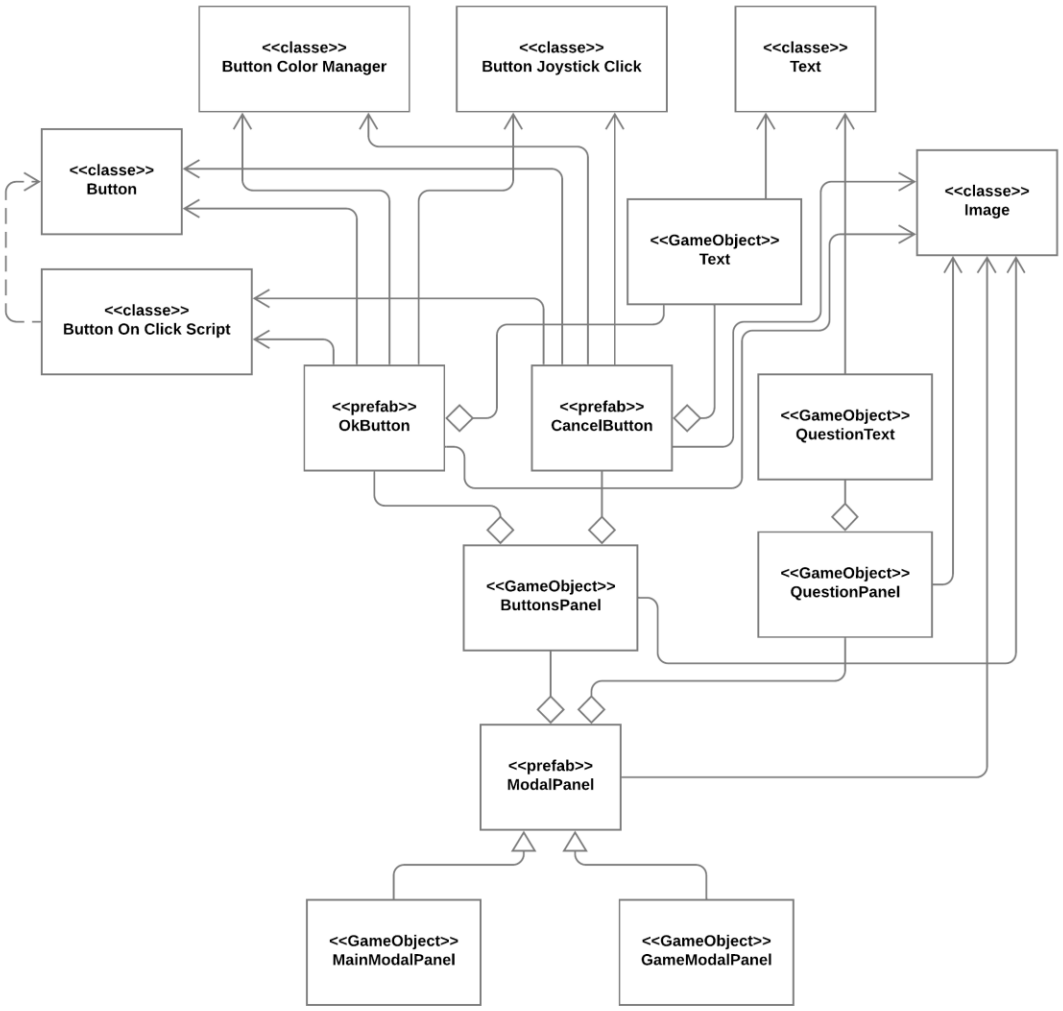
Fonte: Autoria própria.

Figura 38: Diagrama do prefab ListPanel.



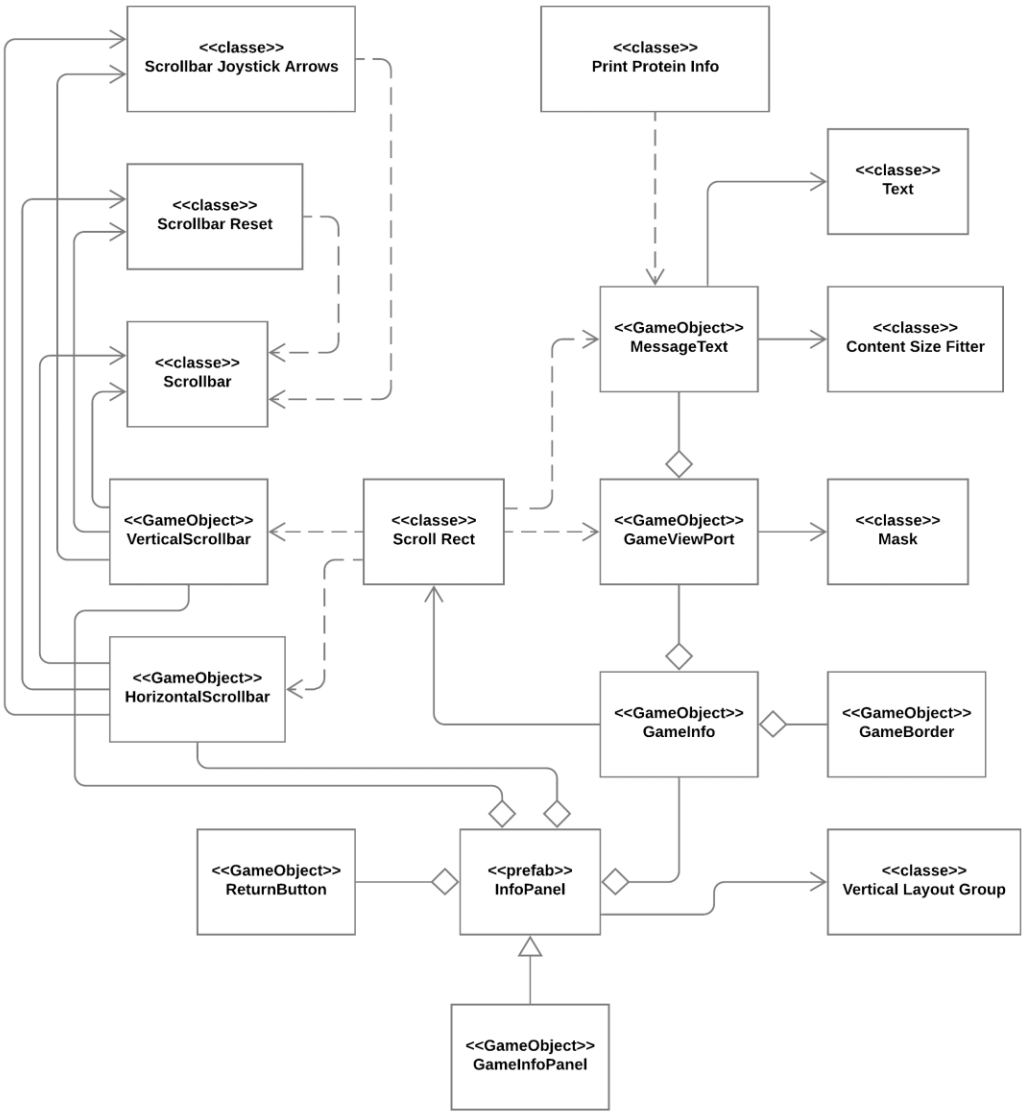
Fonte: Autoria própria.

Figura 39: Diagrama do prefab ModalPanel.



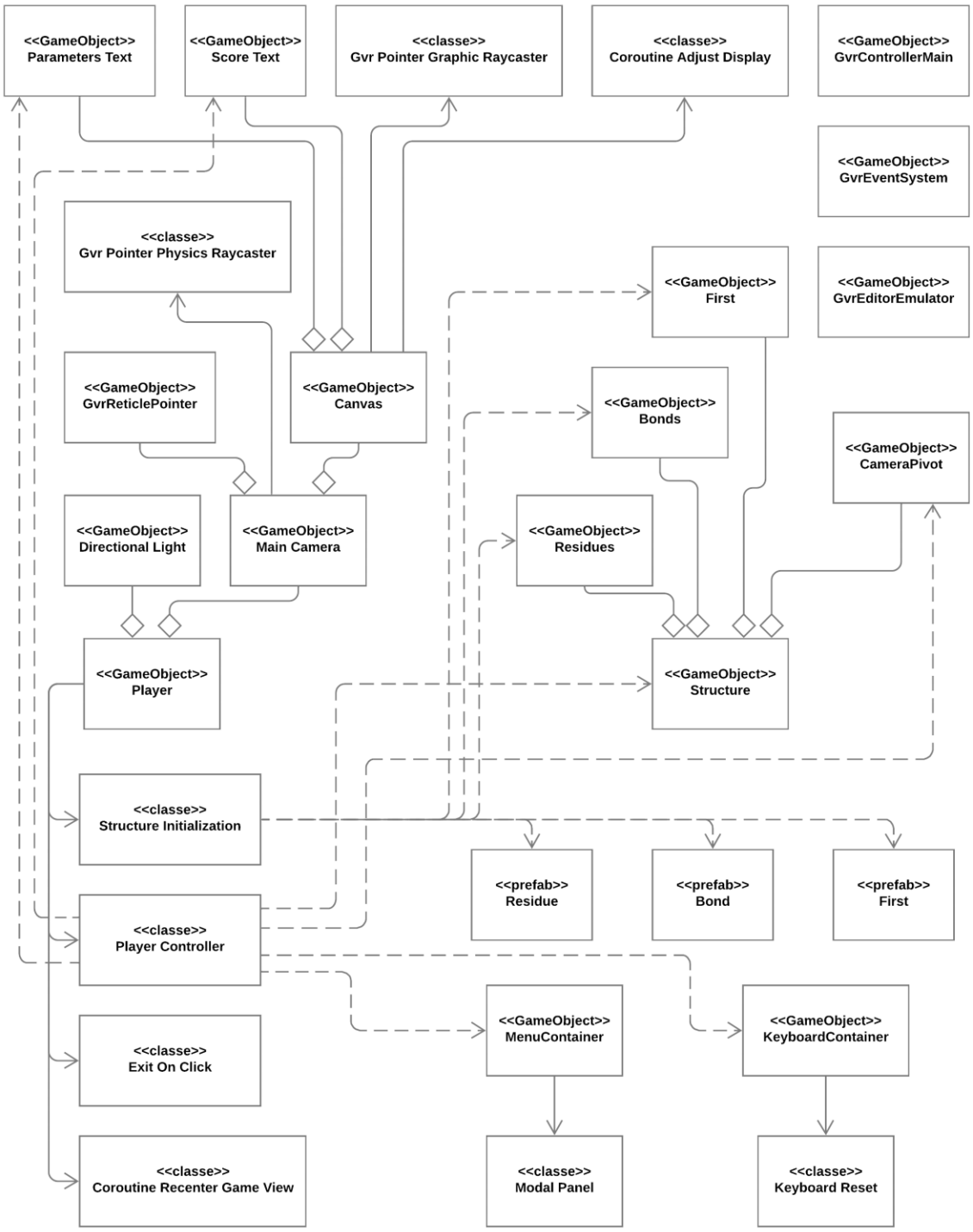
Fonte: Autoria própria.

Figura 40: Diagrama do prefab InfoPanel.



Fonte: Autoria própria.

Figura 41: Diagrama do ambiente de jogo (Game Scene).



Fonte: Autoria própria.