

UNIVERSIDADE PAULISTA – UNIP

**GABRIEL DE ALMEIDA BATISTA
FELIPE DA SILVA BORGES NEVES
FELIPE CUNHA SANTOS
JOSÉ VITOR ZANONI DA COSTA**

**DESENVOLVIMENTO DE UM JOGO:
UTILIZANDO INTERFACE GRÁFICA**

**SÃO PAULO
2016**

**GABRIEL DE ALMEIDA BATISTA
FELIPE DA SILVA BORGES NEVES
FELIPE CUNHA SANTOS
JOSÉ VITOR ZANONI DA COSTA**

**DESENVOLVIMENTO DE UM JOGO:
UTILIZANDO INTERFACE GRÁFICA**

Atividade prática supervisionada e apresentada ao curso Ciência da Computação, para fins de conhecimento na área.

Orientador: Wellington Barbosa.

**SÃO PAULO
2016**

DEDICATÓRIA

Dedicamos este trabalho primeiramente a Deus por ter nos dado vida até este presente momento e aos nossos pais que nos educaram, com força e amor incondicionais.

AGRADECIMENTOS

Agradecemos em primeiro lugar a Deus por ser a base das nossas conquistas.

Aos nossos pais, por acreditarem e terem interesse em nossas escolhas, apoiando-nos e esforçando-se junto a nós, para que supríssemos todas elas.

Ao professor Wellington Barbosa, pela dedicação em suas orientações prestadas na elaboração deste trabalho, nos incentivando e colaborando no desenvolvimento de nossas ideias.

*“Os melhores livros são aqueles que aqueles
que leem acreditam que teriam podido
escrever”.*

(Blaise Pascal)

RESUMO

Um game é uma atividade lúdica composta por uma série de ações e decisões, limitada por regras e pelo universo do game, que resultam em uma condição final. As regras do universo do game são apresentadas por meios eletrônicos controlados por um programa digital. Os jogos digitais, muito conhecidos como games, fazem parte da cultura de massa há pelo menos 50 anos. A maioria dos games é criada para um segmento específico de consumidores, não para o público em geral. O objetivo deste trabalho é avaliar de forma clara o conceito de jogos, sua utilização e sua aplicabilidade, por meio de um jogo 2D desenvolvido na linguagem Java. O trabalho realizado teve caráter qualitativo. Fundamentado em obras publicadas por estudiosos especialistas, que já apontaram os conceitos, usos, aplicações, desempenho e falhas de certos jogos digitais. Nos últimos anos, um novo gênero emergiu na indústria de games, chamado de games casuais. O número de games crescia cada vez mais, incluindo não só fanáticos como também um número cada vez maior de pessoas comuns.

Palavras-chave: jogos digitais; jogo 2D; linguagem Java; banco Oracle.

ABSTRACT

A game is a recreational activity consists of a series of actions and decisions, limited by rules and the game's universe, resulting in a final condition. The game of the universe rules is presented electronically controlled by a digital program. Digital games, games as well known, are part of the mass culture for at least 50 years. Most games are created for a specific segment of consumers, not the general public. The objective of this study is to evaluate clearly the concept of games, their use and its applicability through a 2D game developed in Java. The work was qualitative. Based on works published by expert scholars who have pointed out the concepts, uses, applications, performance and failures of certain digital games. In recent years, a new genre emerged in the games industry, called casual games. The game number grew increasingly including not only fanatics as well as a growing number of ordinary people.

Keywords: digitais games; 2D game; Java language; Oracle database.

LISTA DE ILUSTRAÇÕES

Imagem 01 – DER do banco de dados.	Pag. 30.
Imagem 02 – MER do banco de dados.	Pag. 30.
Imagem 03 – Diagrama do Menu.	Pag. 36.
Imagem 04 – Diagrama de Colisão do Jogador.	Pag. 37.
Imagem 05 – Diagrama de Colisão do Inimigo.	Pag. 38.
Imagem 06 – Diagrama de Movimento do Jogador.	Pag. 39.
Imagem 07 – Glaucus Atlanticus.	Pag. 41.
Imagem 08 – Tardigradae Turritopsis nutricula.	Pag. 42.
Imagem 09 – Esboço do Personagem.	Pag. 43.
Imagem 10 – Esboço do Vermillion.	Pag. 44.
Imagem 11 – Esboço do Raius.	Pag. 44.
Imagem 12 – Esboço do Eleptrus.	Pag. 45.
Imagem 13 – Esboço do Iminumus.	Pag. 45.
Imagem 14 – Esboço do Hardron.	Pag. 46.
Imagem 15 – Ameba.	Pag. 49.
Imagem 16 – Amoeba.	Pag. 49.
Imagem 17 – Fase 02.	Pag. 50.
Imagem 18 – Hardron.	Pag. 50.
Imagem 19 – Fim de Jogo.	Pag. 50.
Imagem 20 – Gota.	Pag. 51.
Imagem 21 – Menu.	Pag. 51.
Imagem 22 – Amoeba Venceu.	Pag. 51.
Imagem 23 – Bem-Vindo.	Pag. 57.
Imagem 24 – Fase-01.	Pag. 57.
Imagem 25 – Fase-03.	Pag. 58.
Imagem 26 – Fase-04.	Pag. 58.
Imagem 27 – Boss.	Pag. 59.
Imagem 28 – Batalha.	Pag. 59.
Imagem 29 – Gabriel de Almeida Batista.	Pag. 67.
Imagem 30 – Felipe da Silva Borges Neves.	Pag. 68.
Imagem 31 – Felipe Cunha Santos.	Pag. 69.
Imagem 32 – José Vitor Zanoni da Costa.	Pag. 70.

LISTA DE TABELAS E GRÁFICOS

Tabela 01 – DD + Trigramação Jogadores.	Pag. 31.
Tabela 02 – DD + Trigramação Recordes.	Pag. 31.
Tabela 03 – Tabela Recorde.	Pag. 33.
Tabela 04 – Tabela Chaves Candidatas.	Pag. 34.

LISTA DE ABREVIATURA E SIGLAS

API – *Applications Programming Interface*.
AWT – *Abstract Window Toolkit*.
CLI – *Comand Line Interface*.
DBA – Administrador de Banco de Dados.
DD – Dicionário de Dados.
DDL – *Data DefinitionLanguage*.
DER – Diagrama de Entidade e Relacionamento.
DML – *Data ManipulationLanguage*.
ER – Entidade e Relacionamento.
FN – Forma Normal.
GB – *Giga Byte*.
GUI – *Graphic User Interface*.
HD – *Hard Disk Drive*.
IDE – *Integrated Development Environment*.
IHC – Interação Humano Computador.
IO – *Input e Output*.
JDBC – *Java DatabaseConnectivity*.
JDK – *Java Developer's Kit*.
JRE – *Java Runtime Environment*.
JSE – *Java Standard Edition*.
MER – Modelo de Entidade e Relacionamento.
TB – *Tera Byte*.
PL/SQL – *Procedural Language/Structured Query Language*.
Pro – Profissional.
RAM – *Random Access Memory*.
SDL – *StorageDefinitionLanguage*.
SGBD – Sistema Gerenciador de Banco de Dados.
SQL – *Structured Query Language*.
URL – *Uniform Resource Locator*.
VDL – *Vision DefinitionLanguage*.

LISTA DE SÍMBOLOS

@ – Arroba.

® – Marca registrada.

™ – *Trade Mark*.

SUMÁRIO

1	INTRODUÇÃO.....	25
2	JOGOS DIGITAIS.....	27
2.1	O mercado.....	26
2.2	Games casuais.....	26
2.2.1	<i>Os jogadores casuais</i>	<i>27</i>
3	PLANO DE DESENVOLVIMENTO	29
3.1	Banco de Dados.....	29
3.1.1	<i>DER.....</i>	<i>30</i>
3.1.2	<i>MER.....</i>	<i>30</i>
3.1.3	<i>DD + Trigramação</i>	<i>31</i>
3.1.4	<i>Classificação de Dados</i>	<i>31</i>
3.1.5	<i>Primeira Forma Normal</i>	<i>32</i>
3.1.6	<i>Segunda Forma Normal</i>	<i>32</i>
3.1.7	<i>Terceira Forma Normal.....</i>	<i>32</i>
3.1.8	<i>Forma Normal de Boyce-Codd</i>	<i>33</i>
3.2	Linguagem de Programação.....	34
3.2.1	<i>Interfaces.....</i>	<i>34</i>
3.2.2	<i>Classes Genéricas.....</i>	<i>35</i>
3.2.3	<i>Métodos Genéricos.....</i>	<i>35</i>
3.3	Diagrama de Bloco	36
3.3.1	<i>Diagrama do Menu</i>	<i>36</i>
3.3.2	<i>Diagrama de Colisão do Jogador</i>	<i>37</i>
3.3.3	<i>Diagrama de Colisão do Inimigo.....</i>	<i>38</i>
3.3.4	<i>Diagrama de Movimento do Jogador.....</i>	<i>39</i>
3.4	Conceito Geral do Game.....	39
3.4.1	<i>Enredo.....</i>	<i>40</i>
3.4.2	<i>Personagem Principal.....</i>	<i>40</i>
3.4.3	<i>Inimigos</i>	<i>43</i>
3.4.4	<i>Ambiente</i>	<i>46</i>
4	DESENVOLVIMENTO.....	47
4.1	Interface Gráfica (GUI).....	47
4.2	Banco de Dados.....	48
4.3	Imagens	48
4.4	Músicas	52
4.5	Armas	52
4.6	Fases	53
4.7	Inimigos.....	55
4.8	Personagem	55
5	RESULTADOS	57
6	CONCLUSÃO.....	61
	REFERÊNCIA BIBLIOGRÁFICA	63
7	FICHAS DE ATIVIDADES PRÁTICAS SUPERVISIONADAS	65

ANEXOS.....	69
--------------------	-----------

1 INTRODUÇÃO

Os jogos digitais, muito conhecidos como games, fazem parte da cultura de massa há pelo menos 50 anos.

Lançado em 1977, o console que seria mais tarde chamado de Atari 2600 chegou a vender oito milhões de unidades até 1983. No início da década de 80, o Atari já era um grande sucesso. O mercado então é inundado por centenas de novos jogos. Acessórios e periféricos revolucionários prometiam ao usuário controlar as ações do jogo apenas com o seu pensamento, como no caso raríssimo do *MindLink*.

Segundo Schuytema (2008, p. 07), um game é uma atividade lúdica composta por uma série de ações e decisões, limitada por regras e pelo universo do game, que resultam em uma condição final.

A indústria de games é impulsionada pelos games, não pelos executivos de marketing ou de negócios, o que a diferencia no mundo do entretenimento.

A maioria dos games é criada para um segmento específico de consumidores, não para o público em geral. Os responsáveis pela tomada de decisões nas editoras de games escolhem os projetos com o maior alcance de público possível e, ao mesmo tempo, tentam atingir jogadores eventuais e fanáticos por outros gêneros de jogos.

Nos últimos anos, um novo gênero emergiu na indústria de games, chamado de games casuais.

Um game casual é qualquer game que pode ser jogado em um intervalo de tempo curto e não requer instruções sobre como jogar. Nesse contexto, quase todos os games árcades clássicos criados se encaixam nesta categoria.

Jogadores casuais são profissionais como médicos, advogados, executivos de negócios, desenvolvedores de software, ou seja, qualquer pessoa. Os games casuais atraem pessoas de todas as culturas, classes sociais, sexo, religião, etnia e orientação política.

Devido ao seu uso muito frequente e ao impacto que os jogos causam, é importante ter a disposição jogos casuais simples, bons, “inocentes”, “puros”, com jogabilidade boa e que divirta o usuário.

A partir destas considerações, visa-se responder a seguinte pergunta: como é feito um jogo digital?

Partindo-se da hipótese, que assimilando o tempo e o conteúdo de estudo com as informações obtidas através de pesquisas é possível, de forma estratégica, desenvolver um jogo 2D na linguagem Java.

O presente trabalho busca contribuir para a sociedade acadêmica mostrando um jogo funcional, seu plano de desenvolvimento e o desenvolvimento em si.

Com pesquisas, avaliações e benefícios, o trabalho busca relatar o processo de desenvolvimento de um jogo digital, desde a sua concepção até a sua finalização.

O trabalho intenciona esclarecer conceitos, funções e aplicações dos jogos casuais. Para tanto, apresentando o desenvolvimento de um game simples com a integração de um banco de dados pequeno.

Fundamentado em obras publicadas por estudiosos especialistas que já apontaram os conceitos, usos, aplicações e falhas no processo de desenvolvimento de jogos digitais. A presente pesquisa busca unificar e transmitir os principais conceitos de suas obras. A junção de trinta e nove obras, de ano e autores diferentes, resulta no presente trabalho.

No primeiro capítulo reunimos conceitos gerais de jogos digitais, história, mercado e tipos de jogadores. Nesta etapa consideramos as definições de dois autores dos que foram consultados, a saber, Harbour (2010), Schuytema (2008) e Vicente Gosciola (2003).

No segundo capítulo é descrito, parcialmente, o planejamento de desenvolvimento do jogo digital. Nesta etapa consideramos as instruções de vinte e nove autores que foram consultados.

O terceiro capítulo é descrito, parcialmente, o processo de desenvolvimento do jogo. Nesta fase consideramos as instruções de nove dos que foram consultados.

O quarto capítulo apresenta algumas imagens do jogo em funcionamento.

2 JOGOS DIGITAIS

Os jogos digitais, muito conhecidos como games, fazem parte da cultura de massa há pelo menos 50 anos.

Lançado em 1977, o console que seria mais tarde chamado de *Atari 2600* chegou a vender oito milhões de unidades até 1983. No início da década de 80, o *Atari* já era um grande sucesso. O mercado então é inundado por centenas de novos jogos. Acessórios e periféricos revolucionários prometiam ao usuário controlar as ações do jogo apenas com o seu pensamento, como no caso raríssimo do *MindLink*.

Harbour (2010, p. 07) evidencia que apesar do grande sucesso comercial do *Atari 2600*, os jogos digitais não ficaram restritos apenas aos consoles de videogame e foram classificados por gêneros. Os games dividem-se em três grandes tipos, reunidos a partir do suporte utilizado: jogos para consoles ocorrem em um console acoplado ao monitor de uma televisão à parte, os jogos para computador são executados no monitor do computador a partir de seu próprio *hardware* e os jogos para arcades.

Como podemos ver, a partir da década de 1970, os games por intermédio dos consoles de videogame passaram a fazer parte da cultura de massa. Assim sendo, como podemos definir um game?

Um game é uma atividade lúdica composta por uma série de ações e decisões, limitada por regras e pelo universo do game, que resultam em uma condição final. As regras do universo do game são apresentadas por meios eletrônicos controlados por um programa digital. As regras e o universo do game existem para proporcionar uma estrutura e um contexto para as ações de um jogador. As regras também existem para criar situações interessantes com o objetivo de desafiar e se contrapor ao jogador. [...]. A riqueza do contexto, o desafio, a emoção e a diversão da jornada de um jogador, e não simplesmente a obtenção da condição final, é que determinam o sucesso do game. (SCHUYTEMA, 2008, p. 07).

Schuytema (2008) apresenta uma definição de quem está inserido na indústria dos games. Porém, questões relativas à definição de jogo estão presentes na literatura há muito tempo. Por este motivo, a ideia de criar um conjunto de definições de jogo em um painel único parece um caminho sensato.

Vicente Gosciola, em entrevista concedida ao projeto Profissão Gamer5 em 2005, afirma que o game é uma hipermídia por excelência. Mas o que é a hipermídia? Antes da era digital, os suportes estavam separados por serem incompatíveis: o desenho, a pintura e as gravuras nas telas, o texto e as imagens no papel, a fotografia e o filme na película química, o som e o vídeo na fita magnética. Depois da passagem pela digitalização, todos esses campos tradicionais de produção de linguagem e processos de comunicação humanos juntaram-se na constituição da hipermídia.

2.1 O mercado

A indústria de games é impulsionada pelos games, não pelos executivos de marketing ou de negócios, o que a diferencia no mundo do entretenimento.

A maioria dos games é criada para um segmento específico de consumidores, não para o público em geral. Os responsáveis pela tomada de decisões nas editoras de games escolhem os projetos com o maior alcance de público possível e, ao mesmo tempo, tentam atingir jogadores eventuais e fanáticos por outros gêneros de jogos. Por exemplo, a *Blizzard Entertainment* preocupa-se exclusivamente com dois gêneros de games: games de estratégias em tempo real – os RTS – e games de interpretação de papéis, os *RPGs*.

Harbour (2010, p. 09) explica que a indústria de games é, na verdade, um mercado de compradores (de jogadores). Ela não é feita de regras ditadas pelos diretores de uma ou de outra empresa, ou pelo pessoal de marketing que ficou intimidado pela recusa dos games em serem influenciados pelas tradicionais teorias de promoção. Em outras palavras, os jogadores são um público muito difícil!

2.2 Games casuais

Harbour (2010, p. 11) promove que nos últimos anos, um novo gênero emergiu na indústria de games, chamado de games casuais. O número de games cresce cada vez mais, incluindo não só fanáticos como também um número cada vez maior de pessoas comuns. Um jogador comum joga algumas horas por semana, enquanto um jogador fanático passa 20 horas ou mais por semana jogando. Os jogadores casuais, por outro lado, passarão apenas alguns minutos jogando um

game de vez em quando. O jogador casual não fica viciado em game algum, como acontece com um fanático.

Sendo assim, o que vem a ser um game casual? Harbour (2010, p. 11) afirma que um game casual é qualquer game que pode ser jogado em um intervalo de tempo curto e não requer instruções sobre como jogar. Nesse contexto, quase todos os games árcades clássicos criados se encaixam nesta categoria.

O mercado de games casuais era limitado há alguns anos. Só recentemente esse assunto começou a aparecer no radar das editoras, escolas e lojas de departamentos, embora os jogadores já brincassem com esses games há duas décadas. Os games casuais são bons para todo mundo, porque são tão fáceis de criar quanto de jogar. (HARBOUR, 2010, p. 10).

2.2.1 *Os jogadores casuais*

Harbour (2010, p. 11) reconhece que os jogadores casuais são profissionais como médicos, advogados, executivos de negócios, desenvolvedores de software, ou seja, qualquer pessoa. Os games casuais atraem pessoas de todas as culturas, classes sociais, sexo, religião, etnia e orientação política.

3 PLANO DE DESENVOLVIMENTO

No plano de desenvolvimento do jogo foi utilizado um laptop da fabricante Samsung com processador *Intel® Core™ i5 – 3230M CPU @ 2.60 GHz*, RAM de 8.00 GB, sistema operacional Windows 10 Pro x64 e HD de 1.00 TB. Os principais softwares utilizados no processo foi o Microsoft Word 2016, Microsoft Excel 2016, Microsoft Visio 2016, brModelo, *Toad Data Modeler*, *CacooSamples* e *Sensus Business ProcessModeler*.

Os principais livros utilizados foram “Projeto de Banco de Dados” de Carlos Alberto Heuser e “Sistemas de banco de dados” de Elmasri e Navathe, ambos se encontram na referência bibliográfica.

3.1 Banco de Dados

Foi utilizado o sistema gerenciador de banco de dados Oracle *Database 11g Release dois* para a construção de nosso banco de dados, do qual armazenará os registros de cada jogador e ordenará em ordem decrescente com base na pontuação individual.

O Oracle *Database 11g Release dois* oferece a base para que a TI forneça com sucesso mais informações com melhor qualidade de serviço, reduza o risco de mudança dentro de suas funções e faça um uso mais eficiente dos orçamentos de TI. Ao implantar o Oracle *Database 11g Release dois* como base para a gestão de dados, as organizações podem usar a capacidade total do banco de dados líder mundial para:

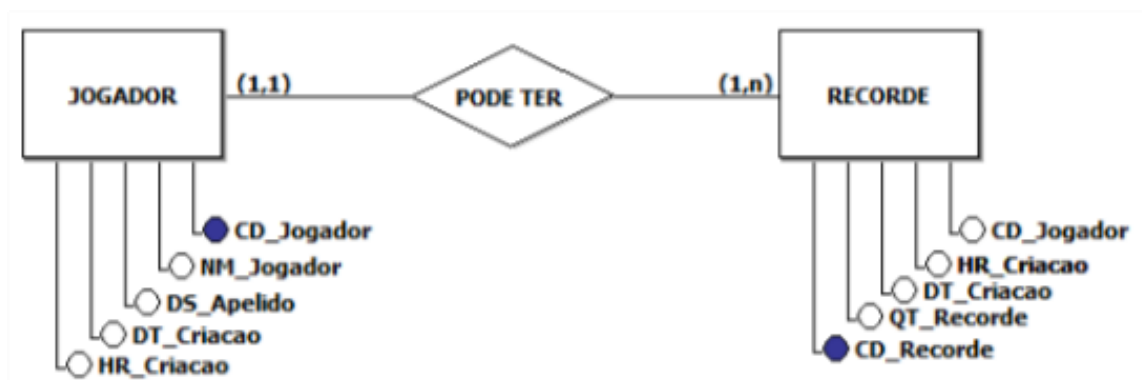
- Reduzir os custos com servidor por um fator de cinco;
- Reduzir os custos com armazenamento por um fator de 12;
- Melhorar o desempenho dos sistemas de missão crítica por um fator de 10;
- Aumentar a produtividade dos DBAs (administradores de banco de dados) por um fator de dois;
- Eliminar a redundância ociosa no centro de dados;
- Simplificar o portfólio geral de software para TI.

Para o planejamento do banco de dados, estivermos levantando a seguinte questão: “um jogador pode ter quantos recordes? Um recorde pode ter quantos jogadores?”. Para replicar este ponto foram empregados o DER, MER, DD + trigramação¹ e as cinco normalizações.

3.1.1 DER

Para a construção do diagrama de entidade e relacionamento foi utilizado o programa brModelo. Segue o resultado:

Imagem 01 – DER do banco de dados.

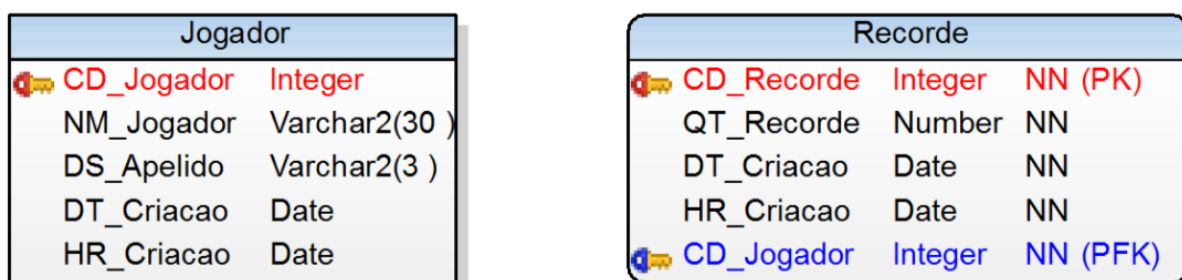


Fonte: Própria, 2016.

3.1.2 MER

Para a construção do modelo de entidade e relacionamento foi utilizado o programa *Toad Data Modeler*. Segue o resultado:

Imagem 02 – MER do banco de dados.



Fonte: Própria, 2016.

3.1.3 DD + Trigramação

Para a construção do dicionário de dados mais trigramação (é uma técnica de banco de dados que se utiliza de prefixos para a identificação de campos conforme suas tabelas) foi utilizado o programa Microsoft Excel 2016. Segue o resultado:

Tabela 01 – DD + Trigramação Jogadores.

Jogadores (JOG)			
Campo	TD	TM	CH
JOG_Codigo	+	-	PK
JOG_Nome	A	30	-
JOG_Apelido	A	3	-
JOG_DT_Criacao	D	-	-
JOG_HR_Criacao	T	-	-

Fonte: Própria, 2016.

Tabela 02 – DD + Trigramação Recordes.

Recordes (REC)			
Campo	TD	TM	CH
REC_Codigo	+	-	PK
JOG_Codigo	N	-	FK
REC_Quantidade	N	4	-
REC_DT_Criacao	D	-	-
REC_HR_Criacao	T	-	-

Fonte: Própria, 2016.

3.1.4 Classificação de Dados

TABELA RECORDE

```

CD_TABELA      +
CD_JOGADOR     +
NM_JOGADOR     +
DS_APELIDO     +
DT_CRIACAO     +
HR_CRIACAO     +
{ CD_RECORDE   +
  QT_RECORDE   +
  DT_CRIACAO   +
  HR_CRIACAO}
```

3.1.5 Primeira Forma Normal

TABELA

CD TABELA

CD JOGADOR

NM_JOGADOR

DS_APELIDO

DT_CRIACAO

HR_CRIACAO

RECORDE

CD RECORDE

CD JOGADOR

CD TABELA

QT_RECORDE

DT_CRIACAO

HR_CRIACAO

3.1.6 Segunda Forma Normal

TABELA

CD TABELA

CD JOGADOR

NM_JOGADOR

DS_APELIDO

DT_CRIACAO

HR_CRIACAO

TAB_RECORDE

CD TABELA

CD RECORDE

CD JOGADOR

RECORDE

CD RECORDE

QT_RECORDE

DT_CRIACAO

HR_CRIACAO

3.1.7 Terceira Forma Normal

TABELA

CD TABELA

CD JOGADOR

NM_JOGADOR

DS_APELIDO

DT_CRIACAO

HR_CRIACAO

TAB_RECORDE

CD TABELA

CD RECORDE

CD JOGADOR

RECORDECD_RECORDER

QT_RECORDER

DT_CRIACAO

HR_CRIACAO

JOGADORCD_JOGADORCD_TABELA

NM_JOGADOR

DS_APELIDO

DT_CRIACAO

HR_CRIACAO

3.1.8 *Forma Normal de Boyce-Codd*

Tabela 03 – Tabela Recorde.

TABELA RECORDER
CD_TABELA
CD_JOGADOR
NM_JOGADOR
DS_APELIDO
DT_CRIACAO_JOG
HR_CRIACAO_JOG
CD_RECORDER
QT_RECORDER
DT_CRIACAO_REC
HR_CRIACAO_REC

Fonte: Própria, 2016.

Chaves candidatas: CD_JOGADOR + DT_CRIACAO_JOG + HR_CRIACAO_JOG.

CD_RECORDER + DT_CRIACAO_REC + HR_CRIACAO_REC.

CD_RECORDER.

CD_TABELA.

CD_JOGADOR.

Tabela 04 – Tabela Chaves Candidatas.

Determinante	Dependentes Funcionais
CD_JOGADOR + DT_CRIACAO_JOG + HR_CRIACAO_JOG	Todos os atributos
CD_RECORDE + DT_CRIACAO_REC + HR_CRIACAO_REC	Todos os atributos
CD_JOGADOR	NM_JOGADOR, DS_APELIDO, DT_CRIACAO_JOG, HR_CRIACAO_JOG
CD_TABELA	-----
CD_RECORDE	QT_RECORDE, DT_CRIACAO_REC, HR_CRIACAO_REC

Fonte: Própria, 2016.

Dois cincos determinantes apresentados apenas os três últimos são chaves candidatas.

3.2 Linguagem de Programação

A linguagem de programação utilizada foi o Java oito (JSE oito) destinada ao desktop. Para saber quais API usufruímos, veja o ANEXO A.

3.2.1 Interfaces

As interfaces têm o propósito principal de suprir a ausência de heranças múltiplas oferecendo um mecanismo mais eficiente. Herança múltipla significa que uma classe possuir mais de uma superclasse, ou seja, derivar de duas ou mais classes e herdar os membros não privado de todas elas.

Ao programar uma interface é preciso implementar todos os métodos definidos nesta interface e, por isso, esta classe pode ser tratada como sendo do tipo que a interface representa. Além disso, uma mesma classe pode implementar múltiplas interface e, portanto, poderá assumir diferentes tipos.

Assim como as classes, as interfaces podem conter atributos e métodos. Entretanto, elas não possuem construtores e funcionam, de certo modo, como uma classe abstrata, não podendo ser instanciadas. Só há um modo de instanciar um objeto cujo tipo seja uma interface: utilizar um construtor de uma classe que a tenha implementado.

Uma das diferenças práticas entre classes e interfaces, é que tanto os atributos quanto os métodos de uma interface devem ser públicos. Todo atributo é implicitamente constante e estático.

Em interfaces, todo método é implicitamente abstrato. Métodos abstratos não possuem corpo, mas somente os elementos que compõem a sua assinatura.

Uma interface somente pode derivar de outra interface assim como uma classe só pode derivar de outra classe.

3.2.2 *Classes Genéricas*

As classes genéricas são classes parametrizadas. Toda classe genérica contém uma seção de parâmetros do tipo logo após o seu nome. A sintaxe aplicável a esta seção de parâmetros de tipo é a mesma que se aplica à seção de parâmetros de tipo dos métodos genéricos. Esta seção é delimitada por colchetes angulares e deve conter a declaração de um ou mais parâmetros de tipo a serem utilizados no corpo da classe. Se houver a declaração de múltiplos parâmetros de tipo, eles devem ser separados por vírgulas. Cada parâmetro de tipo é um identificador utilizado para representar um tipo a ser empregado na classe.

3.2.3 *Métodos Genéricos*

Método genérico é uma implementação única de determinado método que pode ser invocada utilizando argumentos de diferentes tipos. Os métodos genéricos contam com a garantia da validade dos tipos empregados. Esta garantia é oferecida em tempo de compilação. Isso significa que cada invocação de um método genérico é avaliada pelo próprio compilador e leva em conta os tipos dos argumentos utilizados.

Todo método genérico contém uma seção de parâmetros de tipo. Esta seção localiza-se antes do tipo de retorno do método e é delimitada por colchetes

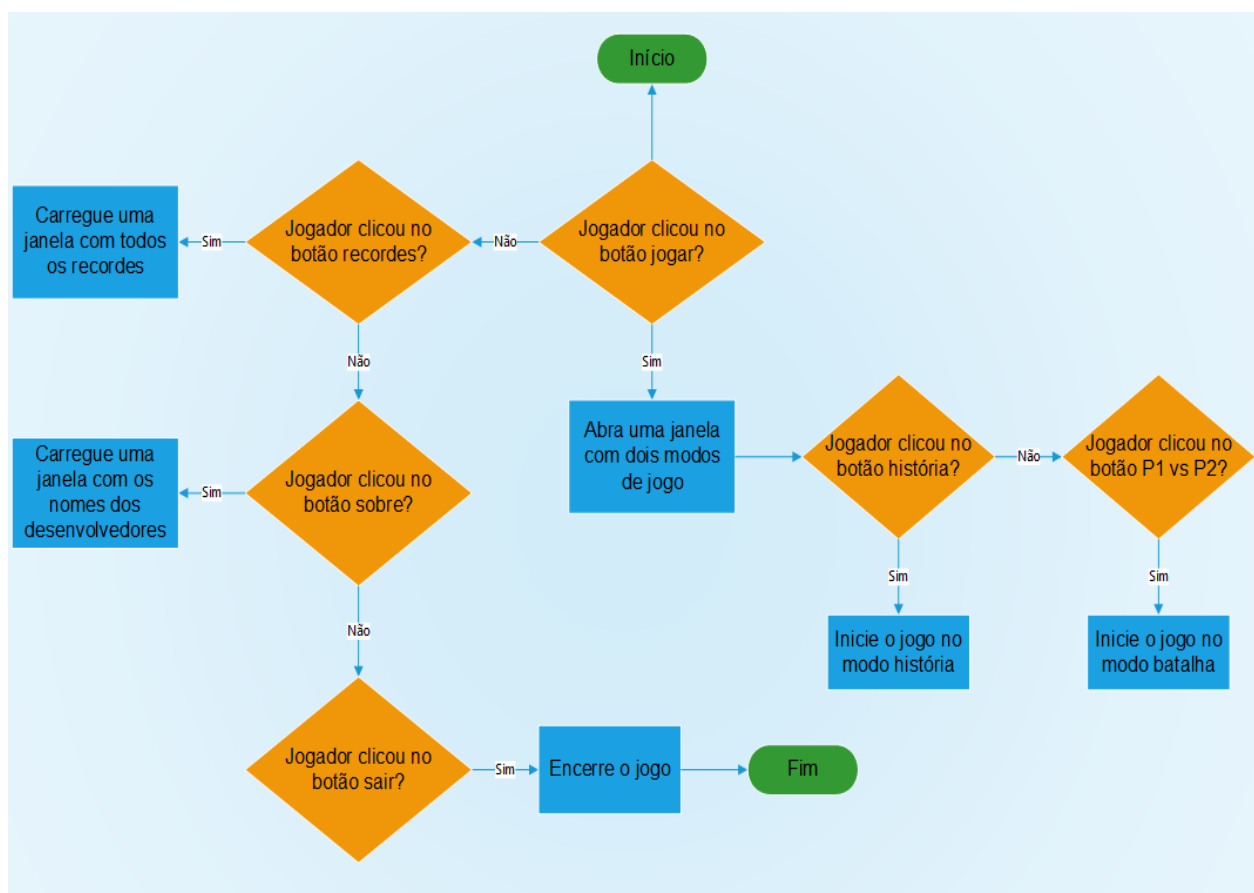
angulares. Ela deve conter a declaração de um ou mais parâmetros de tipo a serem utilizados pelo método genérico. Se houver a declaração de múltiplos parâmetros de tipo, eles devem ser separados por vírgulas.

3.3 Diagrama de Bloco

Para a construção do diagrama de bloco foi utilizado o programa Microsoft Visio 2016.

3.3.1 Diagrama do Menu

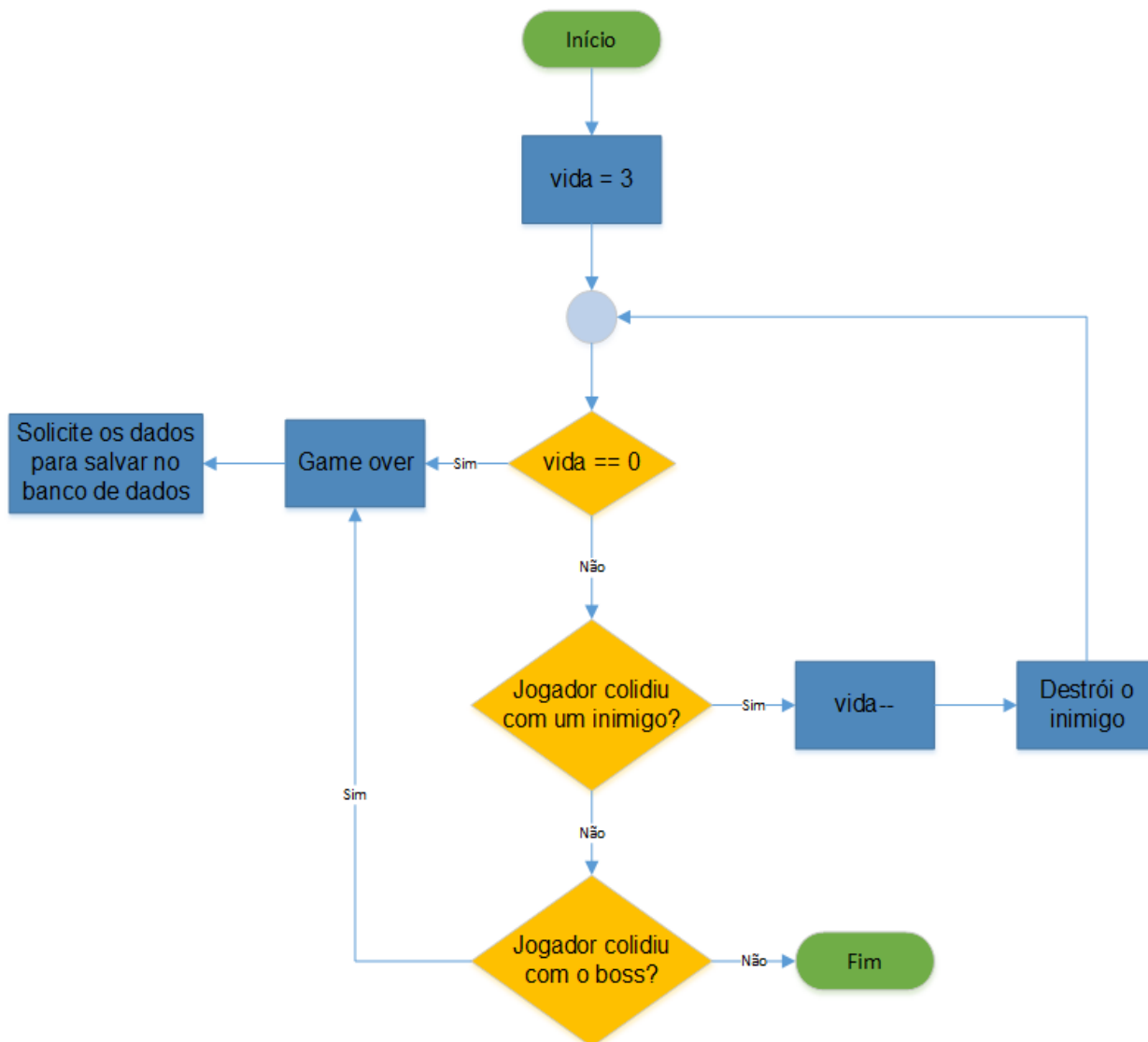
Imagem 03 – Diagrama do Menu.



Fonte: Própria, 2016.

3.3.2 Diagrama de Colisão do Jogador

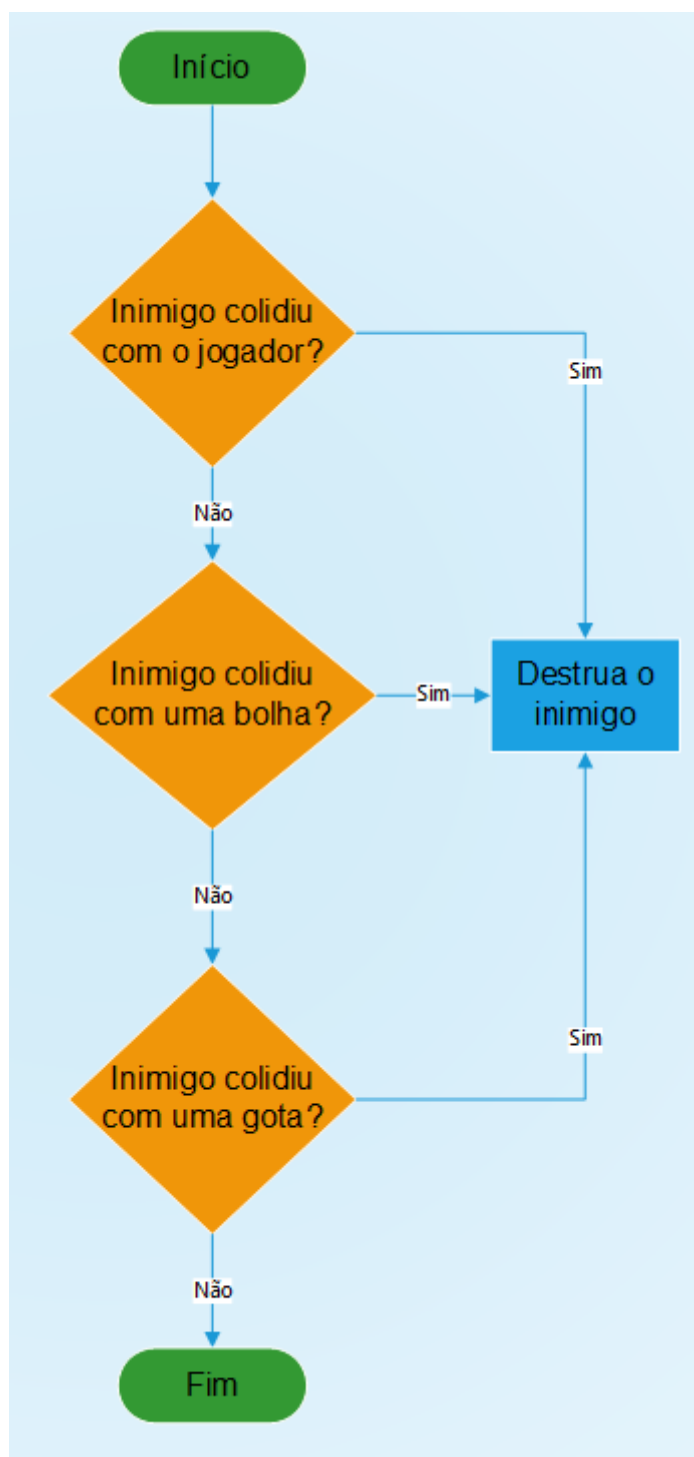
Imagem 04 – Diagrama de Colisão do Jogador.



Fonte: Própria, 2016.

3.3.3 Diagrama de Colisão do Inimigo

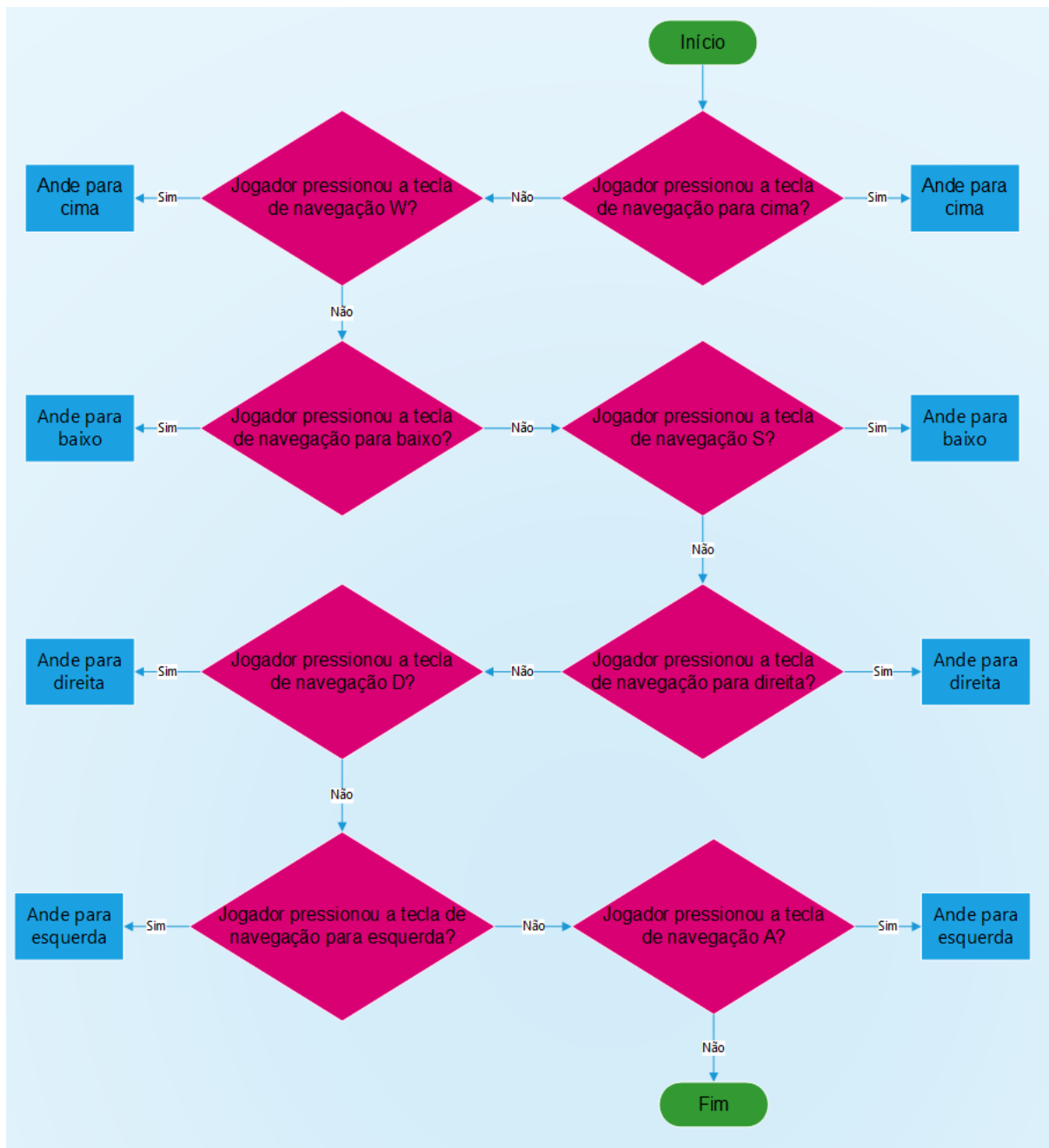
Imagem 05 – Diagrama de Colisão do Inimigo.



Fonte: Própria, 2016.

3.3.4 Diagrama de Movimento do Jogador

Imagem 06 – Diagrama de Movimento do Jogador.



Fonte: Própria, 2016.

3.4 Conceito Geral do Game

O jogo Alpha Gênesis foi inspirado no *Spore* e *Space Invaders*, um jogo de videogame de arcade desenhado por Tomohiro Nishikado e lançado em 1978. O gênero é ação da categoria *Shoot'em up*, um subgênero dos jogos eletrônicos de

tiro. Em um *shoot'em up*, o personagem do jogador se envolve em um ataque solitário, disparando em um grande número de inimigos. Possui gráfico 2D com perspectiva top-down, visão sobre a cabeça do personagem.

3.4.1 *Enredo*

Em novembro do ano 3000, graças a avanços tecno-científicos da bioengenharia microbiana, os seres humanos puderam desenvolver uma artificial forma de vida capaz de se adaptar a ambientes heterogêneos e até se reproduzir. E tentados a uma ambição maior, encapsularam essa bioforma e enviaram-na para um planeta de GZ (*Goldilocks Zone*) semelhante ao da Terra, o GJ1214B.

A bioforma microbiana, batizada de *Glaucus*, foi enviada então ao planeta com a missão de sobreviver, se adaptar e reproduzir nesse novo mundo desconhecido. Para isso, ela conta com mecanismos de defesa, como o disparo de peróxido de hidrogênio com hidroquinona em forma de bolhas, que consequentemente repelem qualquer tipo de inimigo.

3.4.2 *Personagem Principal*

O personagem principal foi inspirado nas características do *Glaucus atlanticus*, uma espécie de lesmas-domar pelágico pertencente ao grupo dos moluscos *nudibrânquios* da família *Glaucidae*, sendo a única espécie conhecida do gênero *Glaucus*. Suas características são: uma coloração branca, azul marinho escuro (azul cobalto), azul claro (azul turquesa) e preto.

Além de possui algumas características do *Glaucus atlanticus*, o personagem principal possui as habilidades do *Tardigrada* e *Turritopsis nutricula*. E seu ataque baseado no besouro-bombardeiro.

Imagem 07 – *Glaucus Atlanticus*.



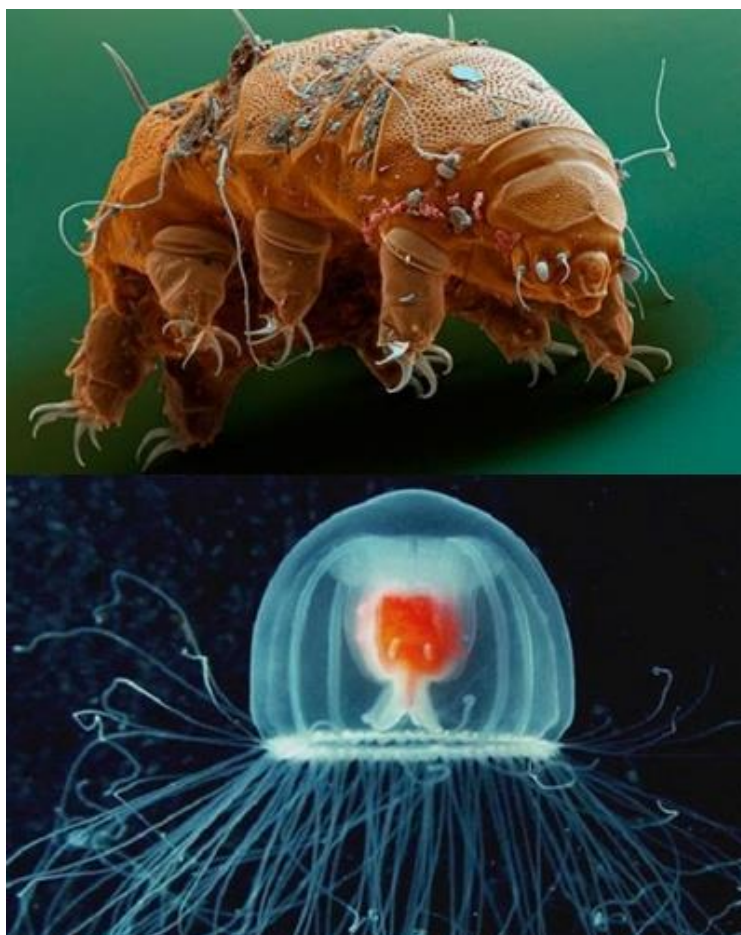
Fonte: *Le Grand Univers*, 2016.

- ***Tardigrada***: também chamado de urso d'água pertence ao Filo *Tardigrada*, tem um tamanho reduzido (0,5 mm a 1,25 mm), possui oito patas, cada uma com quatro a oito pequenas garras, costuma viver entre musgos e líquens, e são fortemente pigmentados. Este ser é recoberto de quitina, possui sistema digestório, se alimenta do conteúdo celular de bactérias ou algas e não possui aparelho respiratório, as trocas gasosas são feitas aleatoriamente com qualquer parte do corpo. São encontrados em todo o planeta inclusive em áreas inóspitas, como a Antártica. Suas principais características são a longevidade e resistência, pode chegar a viver 120 anos, um tempo considerável para um animal de seu tamanho, além disso, tem a capacidade de “desligar” seu metabolismo em condições adversas (como em um intenso inverno ou seca), mas sua principal e espetacular habilidade é a capacidade de reparar o DNA de danos radioativos. Ele pode suportar a uma pressão gigantesca, dezenas de vezes maiores que a enfrentada por seres das zonas abissais. Podem aguentar por alguns minutos uma temperatura de 200°C e ainda suportam uma radiação de 5700 *grays* (um ser humano morre ao ser exposto a 100 *grays*). Algumas experiências foram feitas com esse ser em laboratórios de Universidades Americanas, ele foi congelado a -271°C e reanimado com apenas uma gota d'água. O segredo dos ursos d'água é que eles podem desidratar quase completamente e

entrar num estado de metabolismo pausado. Nesse estado, eles podem ficar “adormecidos” por milhares de anos e voltar a vida com uma simples gota d’água.

- ***Turritopsis nutricula***: é um hidrozoário com um ciclo de vida no qual se reverte ao estágio de pólipo após chegar-se a maturidade sexual. É o único caso conhecido de um animal metazoário capaz de reverter completamente a um estágio de imaturidade sexual, o estágio de colônia após ter alcançado a maturidade sexual é um estágio solitário. Sendo, portanto, o único ser vivo capaz de viver indefinidamente (imortal, caso não haja morte por violência) descoberto pela ciência.

Imagem 08 – *Tardigradae Turritopsis nutricula*.



Fonte: Movimento Jovem, 2016.

Segue um esboço do personagem principal:

Imagem 09 – Esboço do Personagem.



Fonte: Própria, 2016.

O personagem principal possui duas variações de cores, quando o jogador selecione a opção de batalha, a primeira possui as características do *Glaucus* e a segunda possui uma coloração vermelho-escuro, vermelho e preto. Ambas possuem as habilidades dos *Turritopsis nutricula* e *Tardigrada*. *Glaucus* foi apelidado de Ameba (personagem da cor azul) e *Amoeba* (personagem da cor vermelha).

Ele é um personagem controlável pelas teclas WASD ou setas do teclado, possui três vidas. E atuará sobre o ambiente e navegará para todos os lados do mapa. As teclas J e K servirão para disparar gotas e bolhas, respectivamente.

3.4.3 Inimigos

O jogo possui cinco inimigos, contanto o chefe, e cada um detêm um nome e suas próprias características.

1. *Vermillion*: aparece na fase inicial e é uma criatura de tamanho médio da cor vermelha, não possui nenhuma habilidade especial e aparece em grande quantidade. Ela foi inspirada no besouro de veado masculino.
2. *Raius*: aparece na fase dois e é uma criatura de porte médio da cor verde, a sua habilidade especial é a velocidade e aparecem em um número reduzido. Ela foi inspirada na raia.
3. *Eleptrus*: aparece na fase três e é uma criatura de tamanho médio da cor amarela com tentáculos azuis, a sua habilidade é a alta resistência

e aparecem em um número elevado. Ela foi inspirada no besouro tropical.

4. *Iminumus*: aparece na fase cinco e é uma criatura de porte pequeno da cor branca, a sua habilidade é a alta velocidade (superior à de *Raius*) e a mesma resistência do *Eleptrus* e aparecem em grande quantidade. Ela foi inspirada no peixe espada.
5. *Hardron*: é o chefe do jogo, possui cem pontos de vida, tamanho grande, da cor roxa. Ele foi inspirado no besouro Hércules.

Imagem 10 – Esboço do *Vermillion*.



Fonte: Própria, 2016.

Imagem 11 – Esboço do *Raius*.



Fonte: Própria, 2016.

Imagem 12 – Esboço do *Eleptrus*.



Fonte: Própria, 2016.

Imagem 13 – Esboço do *Iminumus*.



Fonte: Própria, 2016.

Imagem 14 – Esboço do *Hardron*.



Fonte: Própria, 2016.

3.4.4 *Ambiente*

O ambiente é naturalista e baseado no do filme *Avatar*. O jogo ocorre no fundo do mar com detalhes únicos de acordo com a vida marinha do planeta Terra, porém voltado para algo mais futurístico e distante de nossa realidade. Ele não deve influenciar na jogabilidade e nem camuflar os inimigos.

4 DESENVOLVIMENTO

No desenvolvimento do jogo foi utilizado um laptop da fabricante Samsung com o processador *Intel®Core™ i5 – 3230M CPU @ 2.60 GHz*, RAM de 8.00 GB, sistema operacional Windows 10 Pro x64 e HD de 1.00 TB e um computador da fabricante Positivo com o *Intel®Celeron®Processor N2808 CPU @ 1.58 GHz*, RAM de 4.00 GB, sistema operacional Linux *Ubuntu x64* e HD de 500 GB. Os principais softwares utilizados no processo foram a IDE Eclipse Java Mars x64 com acesso ao JDK 1.8.0_92 x64 e JRE 1.8.0_102 x64, últimas versões até o presente momento de dissertação, Oracle *Database 11g Express Edition*, *Sublime Text* três, *Visual StudioCode*, *Git*, *Gimp*, *Photoshop CS6*, *Notepad* e o Prompt de Comando.

Os principais livros utilizados foram “Java como programar” de Paul Deitel e Harvey Deitel, “Sistemas de banco de dados” de Elmasri e Navathe, e “Programação de games com Java” de Jonathan S. Harbour, todos se encontram na referência bibliográfica.

A linguagem de programação utilizada foi o Java oito (JSE oito) destinada ao desktop.

4.1 Interface Gráfica (GUI)

No desenvolvimento da interface gráfica foi utilizado o plug-in, da IDE Eclipse, Window Builder. Este plug-in facilita no manuseio das APIs AWT e Swing, dando a possibilidade de usufruir de suas capacidades criativas de desenvolvimento de GUI.

No Java, o desenvolvimento de interface gráfica pode ser realizado com o uso de componentes disponíveis em duas APIs distintas: o AWT e o Swing. Uma GUI é desenvolvida utilizando-se componentes, que são objetos visuais que possibilitam ao usuário realizar a interação com o programa por meio de um ambiente gráfico. (SANTOS, 2014, p. 970).

A grande vantagem do Window Builder é fornecer a possibilidade de “clicar e arrastar” os componentes desejados para a janela, chamada de *Jframe*. Alguns componentes comuns são os rótulos, os botões, os campos de texto, as áreas de texto, as caixas de checagem, os botões de rádio, as listas e os menus. Nomeados,

34 em Java, de JLabel, JButton, JTextField, JTextArea, JCheckBox, JRadioButton, JList e JMenuBar, respectivamente.

4.2 Banco de Dados

Para o desenvolvimento do banco de dados foi utilizado o SGBD Oracle *Database 11g Express Edition*. A linguagem utilizada no Oracle *Database 11g Express Edition* é a *PL/SQL*. Segue um dos códigos utilizados no desenvolvimento do banco.

```
CREATE TABLE "RECORDES"
(
    "REC_CODIGO" NUMBER(10,0) NOT NULL ENABLE,
    "REC_DT_CRIACAO" DATE,
    "REC_QUANTIDADE" NUMBER(10,0) NOT NULL ENABLE,
    "JOG_CODIGO1" NUMBER(10,0),
    PRIMARY KEY ("REC_CODIGO") ENABLE
);
```

```
ALTER TABLE "RECORDES" ADD CONSTRAINT "FKD05CA8BF26D7C632" FOREIGN KEY
("JOG_CODIGO1") REFERENCES "JOGADORES" ("JOG_CODIGO") ENABLE;
```

Inicialmente foi criada a tabela *RECORDES* com quatro atributos, do qual o primeiro é chave primária e os demais são atributos “comuns”. Ela não possui nenhum relacionamento no momento de sua criação.

Foi necessário realizar uma alteração na tabela *RECORDES*, adicionando mais um atributo a mesma do qual seria designado para referência uma chave estrangeira da tabela *JOGADORES*.

4.3 Imagens

A classe *ImageEntity* é responsável por manipular as imagens dos objetos, ela é uma subclasse da *BaseGameEntity*, possui quatro atributos dos quais dois são do tipo referencial e os outros dois do tipo primitivo. Ela usufrui das classes *Graphics2D*, *Image* e *Rectangle*. Os principais métodos são:

```

public void draw(int x, int y) {
    g2d.drawImage(getImage(), x, y, null);
}
public Rectangle getBounds() {
    return new Rectangle((int)getX(), (int)getY(), getWidth(), getHeight());
}

```

O método *draw* recebe as coordenadas, *x* e *y*, onde ele deverá desenhar as imagens usando o método *drawImage* do objeto *Graphics2D* *g2d*. Perceba que o método do objeto *g2d* está recebendo quatro parâmetros – *getImage()*, *x*, *y*, *null* – o primeiro fornece a imagem que ele irá desenhar, o segundo e o terceiro são as coordenadas e o quarto é a imagem que ele irá observar, no nosso caso não fornecemos nenhuma.

O método *getBounds* retorna um objeto do tipo *Rectangle* necessário para a construção das máscaras de cada imagem, que verifica as colisões que ocorreram no jogo.

Os programas *Photoshop* CS6 e *GIMP* foram fundamentais para o renderizamento dos esboços. Segue alguns dos resultados, os demais se encontram nos anexos.

Imagem 15 – Ameba.



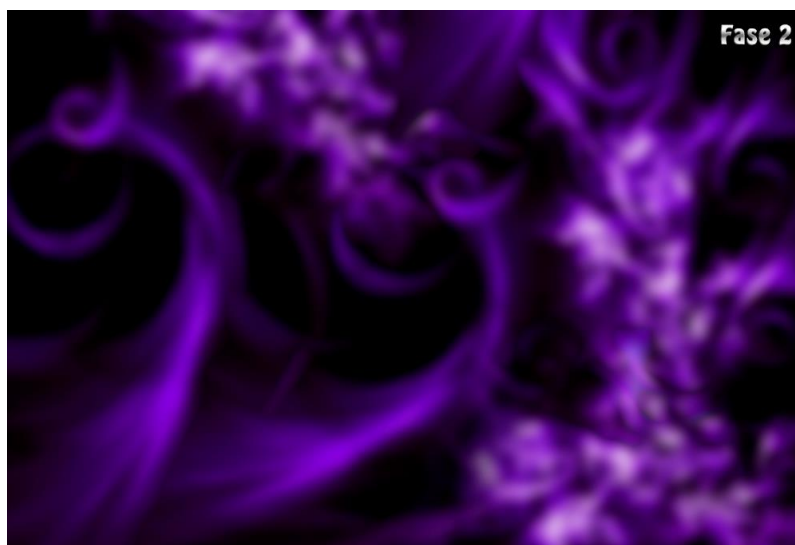
Fonte: Própria, 2016.

Imagem 16 – Amoeba.

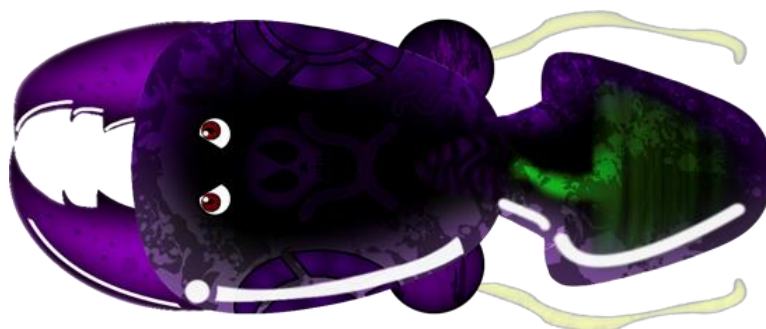


Fonte: Própria, 2016.

Imagem 17 – Fase 02.



Fonte: Própria, 2016.

Imagem 18 – *Hardron*.

Fonte: Própria, 2016.

Imagem 19 – Fim de Jogo.



Fonte: Própria, 2016.

Imagem 20 – Gota.



Fonte: Própria, 2016.

Imagem 21 – Menu.



Fonte: Própria, 2016.

Imagem 22 – Amoeba Venceu.



Fonte: Própria, 2016.

4.4 Músicas

A classe responsável por manipular as músicas do jogo foi nomeada de *TocarSom* e implementa a interface *ActionListener*. Ela usufrui das classes *ActionEvent*, *ActionListener*, *FileInputStream*, *IOException*, *AudioPlayer* e *AudioStream*. O principal método é:

```
public static void som(String musica) {
    AudioPlayer MGP = AudioPlayer.player;
    AudioStream BGM = null;
    try {
        BGM = new AudioStream (new FileInputStream("res/"+musica+".wav"));
    } catch (IOException error) {
        System.out.println("ERRO!");
    }
    MGP.start(BGM);
}
```

O método *som* recebe uma *String* que servirá como parâmetro para o objeto *BGM* responsável por atribuir o caminho da música. Caso o caminho não seja encontrado, o método mostrará uma mensagem de erro. Caso contrário ele chamará o método do objeto *MGP*, responsável por iniciar a música.

Vale ressaltar que a classe *AudioPlayer* só recebe arquivos com a extensão *WAV* e que é uma API precária.

4.5 Armas

As classes *Bolha* e *Gota* são responsáveis por manipular as armas que o personagem principal utilizará, ambas são subclasse da classe *ImageEntity*. Elas usufruem das classes *Random* e *ImageIcon*. O principal método, de ambas, é o construtor:

```

public Construtor(int x, int y) {
    Random rand = new Random();
    ImagemIcon referencia1 = new ImagemIcon("res\\imagem.png");
    ImagemIcon referencia2 = new ImagemIcon("res\\imagem.png");
    ImagemIcon referencia3 = new ImagemIcon("res\\imagem.png");
    ImagemIcon referencia4 = new ImagemIcon("res\\imagem.png");
    setX(x);
    setY(y);
    randomNum = rand.nextInt(i);
    if(randomNum == i)
        setImage(referencia1.getImage());
    else if(randomNum == i)
        setImage(referencia2.getImage());
    else if(randomNum == i)
        setImage(referencia3.getImage());
    else if(randomNum == i)
        setImage(referencia4.getImage());
    else if(randomNum == i)
        setImage(referencia1.getImage());
    else if(randomNum == i)
        setImage(referencia2.getImage());
    setVisivel(true);
}

```

O método construtor recebe as coordenadas e passa as mesmas para os métodos herdados da classe *ImageEntity*. Ele também cria quatro objetos, denominada referência, que são utilizados para localizar as imagens e inseri-las no método *setImage*, também herdado da superclasse. Os *ifs* são usados para desenhar diferentes formas de uma mesma munição com base em um número aleatório retornado pelo método *nextInt*, pertencente ao objeto *rand* do tipo *Random*.

4.6 Fases

O jogo possui cinco fases e cada uma delas possuem uma classe que usufrui das classes *ArrayList*, *List* e *ImagemIcon*. Todas as classes possuem a mesma lógica de controle, mas o fundamental de cada uma é o método que verifica as colisões do jogo. Vejamos:

```

public boolean isColisao(Personagem tipo, List<Bolha>bolhas, List<Gota>gotas) {
    if(tipo instanceof PersonagemBlue) {
        personagem = (PersonagemBlue) tipo;
    }
    else if(tipo instanceof PersonagemRed){
        personagem = (PersonagemRed) tipo;
    }
    bolhas = personagem.getBolhas();
    gotas = personagem.getGotas();
    for (int l = 0; l < sizeInimigo(); l++)
        if(personagem.getBounds().intersects(getInimigo(l).getBounds()))
        {
            personagem.setVisivel(false);
            deletelnimigo(l);
            personagem.decVida(1);
            if(personagem.getVida() == 0)
                return true;
        }
    for (int j = 0; j < bolhas.size() - 1; j++)
        for (int k = 0; k < sizeInimigo(); k++) {

            if(bolhas.get(j).getBounds().intersects(getInimigo(k).getBounds())) {
                deletelnimigo(k);
                bolhas.remove(j);
                Control.incPontos(1);
            }
            else if(bolhas.get(j).getX() > 960)
                bolhas.remove(j);
        }

    for (int k = 0; k < gotas.size() - 1; k++)
        for (int l = 0; l < sizeInimigo(); l++) {
            if(gotas.get(k).getBounds().intersects(getInimigo(l).getBounds())){
                deletelnimigo(l);
                gotas.remove(k);
                Control.incPontos(1);
            } else if(gotas.get(k).getY() > 672)
                gotas.remove(k);
        }
    return false;}

```

O método *ifColisao* recebe o personagem, as bolhas e as gotas que estiverem instanciadas. Ele verifica qual o personagem em jogo, se é o azul ou o vermelho, e depois entra em alguns laços de repetições que verificará se a máscara da imagem de cada objeto está colidindo uma na outra. Caso haja uma intersecção o método remove realiza uma instância e retorna verdadeiro para a classe que o invocou, caso contrário apenas retorna falso.

4.7 Inimigos

O jogo possui cinco inimigos e cada um deles possui uma subclasse da classe *ImageEntity* que usufrui da classe *Imagelcon*. Todas as classes possuem a mesma lógica de controle, mas o fundamental de cada uma é o método que verifica construtor. Vejamos:

```
public Construtor(int x, int y) {
    Imagelcon referencia;
    setX(x);
    setY(y);
    if(55estiári++ % 3 == 0)
        referencia = new Imagelcon("res\\imagem.png");
    else
        referencia = new Imagelcon("res\\imagem.png");
    image = referencia.getImage();
    setVisivel(true);
}
```

Ao instanciar um inimigo, devemos passar para o construtor as coordenadas que cada inimigo iniciará o jogo. O laço de condição serve para capturar imagens de tamanho variado de cada inimigo.

4.8 Personagem

O jogo possui dois personagens diferentes, mas ambos possuem a mesma implementação de interface, herdam da classe mesma classe (*ImageEntity*) e lógica. O principal método é o responsável por movimentar o objeto. Vejamos:

```
public void keyPressed(KeyEvent e) {  
    int codigo = e.getKeyCode();  
    switch(codigo) {  
        case KeyEvent.VK_W:  
            dy = -1;  
            image = anim_U.getImage();  
            break;  
        case KeyEvent.VK_S:  
            dy = 1;  
            image = anim_D.getImage();  
            break;  
        case KeyEvent.VK_A:  
            dx = -1;  
            image = anim_L.getImage();  
            break;  
        case KeyEvent.VK_D:  
            dx = 1;  
            image = anim_R.getImage();  
            break;  
    }  
}
```

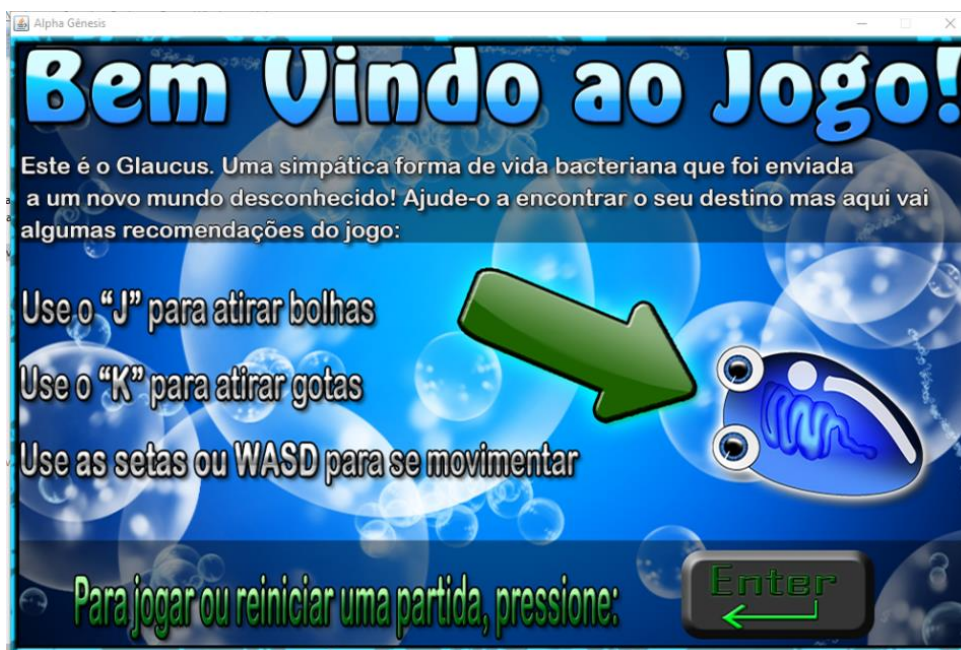
O método *keyPressed* recebe como parâmetro um objeto do tipo *KeyEvent* que captura as teclas que forem pressionadas durante a execução do jogo.

A estrutura de decisão *switch* verifica se a tecla pressionada é uma das que movimentam o personagem ou atiram no jogo.

5 RESULTADOS

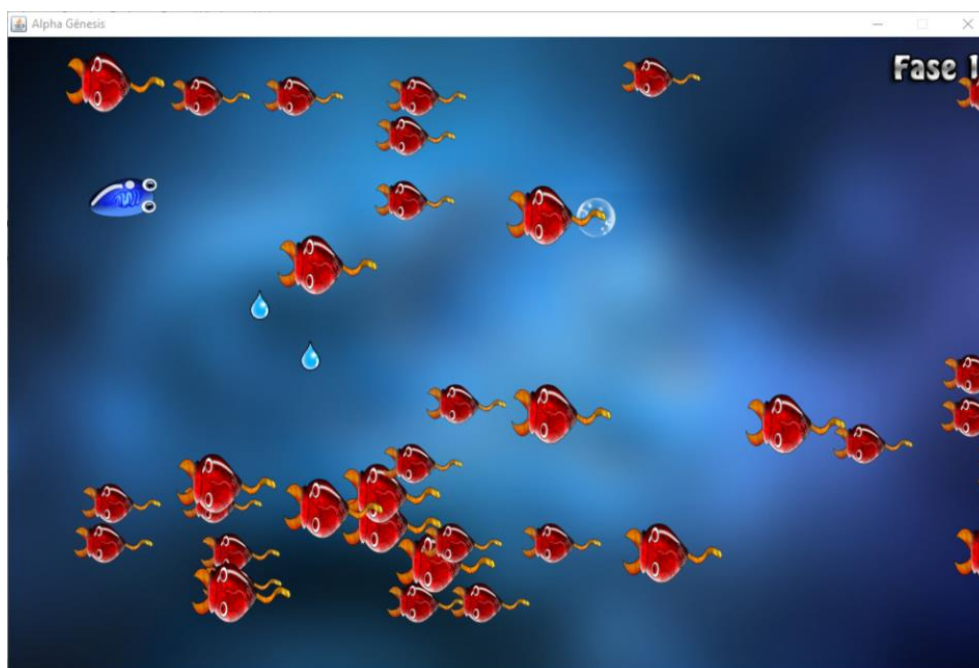
Segue uma lista de Printscreen do jogo em execução.

Imagem 23 – Bem-vindo.



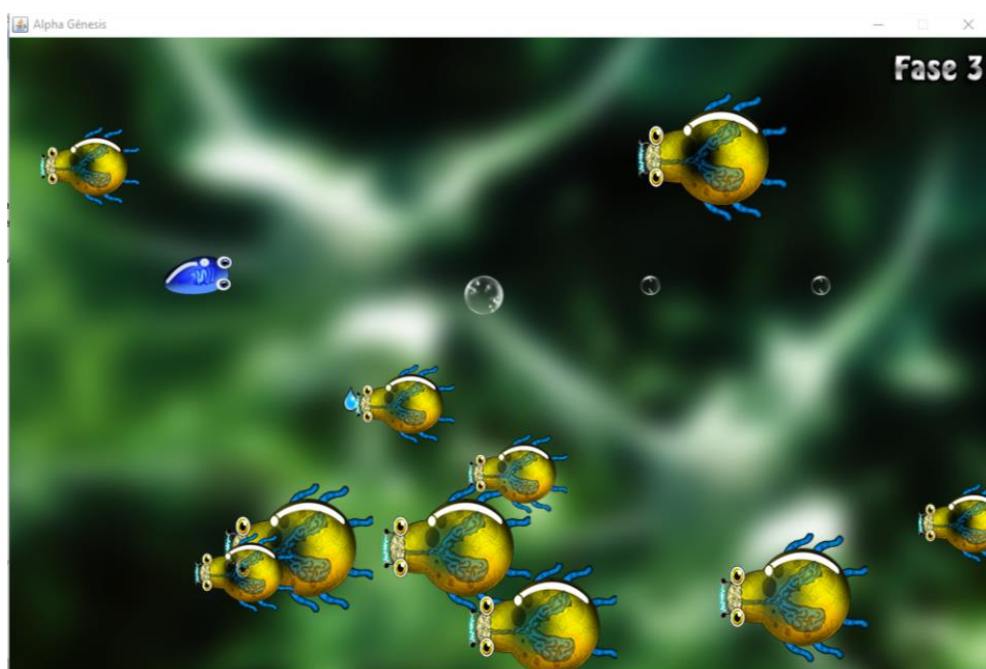
Fonte: Própria, 2016.

Imagem 24 – Fase-01.



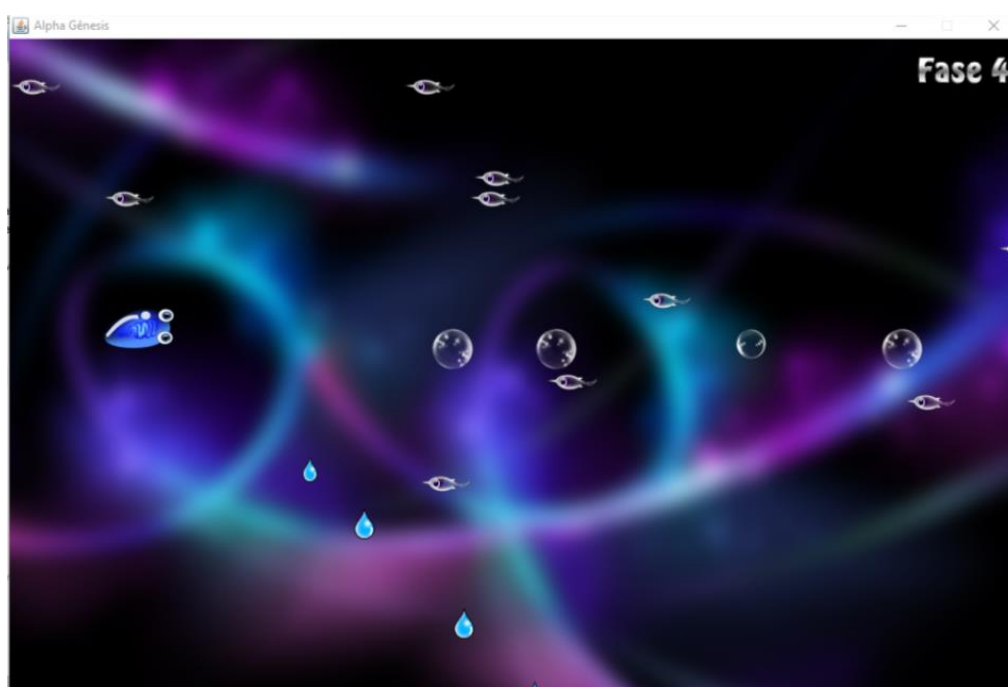
Fonte: Própria, 2016.

Imagem 25 – Fase-03.



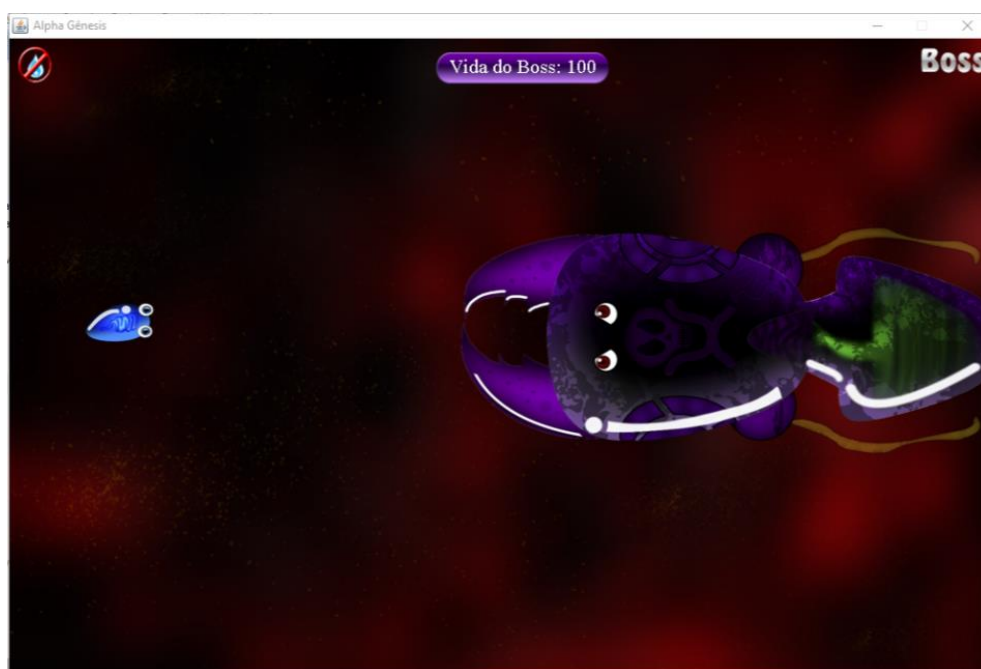
Fonte: Própria, 2016.

Imagem 26 – Fase-04.



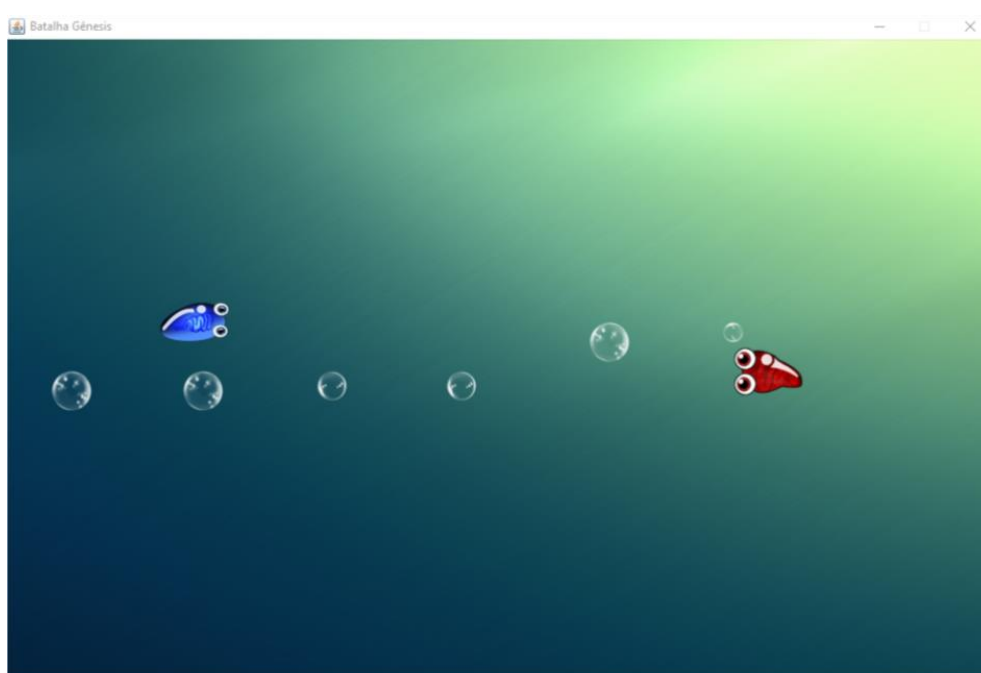
Fonte: Própria, 2016.

Imagem 27 – Boss.



Fonte: Própria, 2016.

Imagem 28 – Batalha.



Fonte: Própria, 2016.

6 CONCLUSÃO

Atualmente o mercado de games movimenta em torno dos US\$ 91.5 bilhões, superando o mercado cinematográfico que movimenta por volta de R\$ 2.25 bilhões.

Ao contrário das outras indústrias de entretenimento, a de games é impulsionada pelo produto em si e não pelos executivos de marketing ou de negócios. Não é obvio que os esportes profissionais (NFL, NBA, NHL e MBA dos Estados Unidos) não sejam impulsionados ou direcionados pelos fãs? Pelo contrário, os fãs geralmente são ridicularizados ou ignorados não apenas pelos patrocinadores das equipes, mas também pelas próprias organizações.

De que forma a indústria de games é um mercado de compradores (ou seja, quando os jogadores têm muita influência sobre os tipos de games que serão criados)? A maioria dos games é criada para um segmento específico de consumidores, não para o público em geral. Os responsáveis pela tomada de decisões nas editoras de games escolhem os projetos com maior alcance de público possível e, ao mesmo tempo, tentam atingir jogadores eventuais e fanáticos por outros gêneros de jogos.

Alguns dos pilares para a construção de um jogo são: programação, modelagem, sonoplastia, roteirismo, plano de negócios e teste. O desenvolvimento do “Alpha Gênesis” englobou programação, modelagem, sonoplastia, roteirismo e teste.

Na parte de programação foi necessário a leitura de alguns livros especializados para a criação de jogos 2D. Foram encontradas algumas dificuldades, como por exemplo, a precariedade de conteúdo em qualquer língua, mesmo sendo utilizada uma das linguagens de programação mais conhecida do mundo, não foi encontrado material de apoio.

O paradigma orientado a objetos, foi um facilitador para o desenvolvimento do game, pois possibilitou uma abstração maior do mundo real.

O framework *Hibernate* possibilitou a persistência de objetos em banco de dados relacionais. Desta forma, não foi necessário a manipulação da linguagem específica para cada sistema de gerenciamento de banco de dados. Capacitando o jogo de manipular dados, independentemente do banco de dados utilizado na máquina.

A modelagem, assim como a programação, é um dos pilares de um desenvolvimento de jogo, pois é ela que cuida da parte gráfica do game, possibilitando um maior nível de entretenimento do game.

Os programas de edições de imagens (Photoshop e Gimp) foram fundamentais para a qualidade visual do game. A maior dificuldade encontrada foi a parte do rascunho, porque foram necessários buscar inspirações de seres não muito conhecidos.

O roteiro do jogo foi inspirado na situação atual do planeta Terra, onde o mesmo se encontra com sérios problemas com a capacidade de aniquilar a espécie humana. Além da situação do planeta Terra, buscamos inspiração na ignorância humana, da qual tende de aumentar a cada geração.

Um jogo sem som não é estimativo, porque a música é arte de manifestar os diversos afetos da nossa alma mediante ao som. O som ele aumenta a capacidade de realidade de um jogo, não importa qual, e também o deixa mais empolgante e prazeroso. Esse é um dos motivos que faz com que a sonoplastia seja um dos pilares.

Concluimos que a criação de jogo não envolve apenas a parte de programação e que ela é uma das, senão a, última a ser iniciada. Um jogo não envolve só a parte de tecnologia, pois ele engloba todas, dependendo do contexto do game, as disciplinas descobertas pelo homem. Por exemplo, para desenvolver um jogo de tiro, precisamos ter preocupações com a física, impacto que irá causar na sociedade, pois é um jogo de violência elevada, marketing, psicologia, filosofia, história (dependendo do enredo do jogo).

Temos exemplos de jogos que não foram planejados corretamente e conseqüentemente foram banidos de alguns países. Por exemplo, Carmagedon (Brasil, Austrália, por ser um jogo onde se ganhava mais pontos por atropelar pedestres), Rape Lay (mundialmente, referencias a estupro), GTA (em diversos países por diversos motivos como músicas, excesso de violência, casos externos que culpavam o jogo e problemas com atores).

REFERÊNCIA BIBLIOGRÁFICA

GOSCIOLA, Vicente. **Roteiro para as novas mídias: do game à TV interativa**. São Paulo, Editora Senac São Paulo, 2003.

HARBOUR, J. S. **Programação de games com Java**. 2. Ed. São Paulo: Cengage Learning, 2010. 417 p.

SCHUYTEMA, P. **Designer de games: uma abordagem prática**. 1. Ed. São Paulo: Cengage Learning, 2008.

7 FICHAS DE ATIVIDADES PRÁTICAS SUPERVISIONADAS

Imagem 29 – Gabriel de Almeida Batista.

UNIP
UNIVERSIDADE PAULISTA

FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: Gabriel de Almeida Batista TURMA: CC4A93 RA: CUU 90F-1

CURSO: Ciência da Computação CAMPUS: Jalisco SEMESTRE: 4º TURNO: Manhã

CÓDIGO DA ATIVIDADE: 985X SEMESTRE: 4º semestre ANO GRADE: 2016

DATA DA ATIVIDADE	DESCRIÇÃO DA ATIVIDADE	TOTAL DE HORAS	ASSINATURA DO ALUNO	HORAS ATRIBUÍDAS (1)	ASSINATURA DO PROFESSOR
01/10	Calota de Aram	3	Gabriel de Almeida		
02/10	Calota de Aram	3	Gabriel de Almeida		
03/10	Calota de Aram	3	Gabriel de Almeida		
04/10	Planejamento de TCC	3	Gabriel de Almeida		
05/10	Planejamento de TCC	3	Gabriel de Almeida		
06/10	Planejamento de TCC	3	Gabriel de Almeida		
07/10	Planejamento de TCC	3	Gabriel de Almeida		
08/10	Planejamento de TCC	3	Gabriel de Almeida		
09/10	Desenvolvimento de TCC	3	Gabriel de Almeida		
10/10	Desenvolvimento de TCC	3	Gabriel de Almeida		
11/10	Desenvolvimento de TCC	3	Gabriel de Almeida		
12/10	Desenvolvimento de TCC	3	Gabriel de Almeida		
13/10	Desenvolvimento de TCC	3	Gabriel de Almeida		
14/10	Desenvolvimento de TCC	3	Gabriel de Almeida		
15/10	Desenvolvimento de TCC	3	Gabriel de Almeida		
16/10	Definição de TCC	3	Gabriel de Almeida		
17/10	Definição de TCC	3	Gabriel de Almeida		
18/10	Definição de TCC	3	Gabriel de Almeida		
19/10	Definição de TCC	3	Gabriel de Almeida		
20/10	Definição de TCC	3	Gabriel de Almeida		

(1) Horas atribuídas de acordo com o regulamento das Atividades Práticas Supervisionadas do curso.

TOTAL DE HORAS ATRIBUÍDAS: 90 horas

AVALIAÇÃO: _____
Aprovado ou Reprovado

NOTA: _____

DATA: ____/____/____

CARIMBO E ASSINATURA DO COORDENADOR DO CURSO

Fonte: Própria, 2016.

Imagem 30 – Felipe da Silva Borges Neves.


UNIP
UNIVERSIDADE PAULISTA

FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: Felipe da Silva Borges Neves TURMA: 04493 RA: 049770-3

CURSO: Curso de Computação CAMPUS: Itatiba SEMESTRE: 4º TURNO: Manhã

CÓDIGO DA ATIVIDADE: 38 FV SEMESTRE: 4º ANO GRADE: 2016

DATA DA ATIVIDADE	DESCRIÇÃO DA ATIVIDADE	TOTAL DE HORAS	ASSINATURA DO ALUNO	HORAS ATRIBUÍDAS (1)	ASSINATURA DO PROFESSOR
1/10	Seleção de dados	3	Felipe Borges		
2/10	Seleção de dados	3	Felipe Borges		
3/10	Planejamento	3	Felipe Borges		
4/10	Planejamento	3	Felipe Borges		
5/10	Planejamento	3	Felipe Borges		
6/10	Planejamento	3	Felipe Borges		
7/10	Planejamento	3	Felipe Borges		
8/10	Planejamento	3	Felipe Borges		
9/10	Desenvolvimento	3	Felipe Borges		
10/10	Desenvolvimento	3	Felipe Borges		
11/10	Desenvolvimento	3	Felipe Borges		
12/10	Desenvolvimento	3	Felipe Borges		
13/10	Desenvolvimento	3	Felipe Borges		
14/10	Desenvolvimento	3	Felipe Borges		
15/10	Teste de aceitação	3	Felipe Borges		
16/10	Teste de aceitação	3	Felipe Borges		

(1) Horas atribuídas de acordo com o regulamento das Atividades Práticas Supervisionadas do curso.

TOTAL DE HORAS ATRIBUÍDAS: 60 horas

AValiação: _____
Aprovado ou Reprovado

NOTA: _____

DATA: / /

CARIMBO E ASSINATURA DO COORDENADOR DO CURSO

Fonte: Própria, 2016.

Imagem 31 – Felipe Cunha Santos.


UNIP
UNIVERSIDADE PAULISTA

FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: Felipe Cunha Santos TURMA: CC3A33 RA: C4695F-2

CURSO: Ciência da Computação CAMPUS: Itatuba SEMESTRE: 3º TURNO: manhã

CÓDIGO DA ATIVIDADE: 385X SEMESTRE: 4º ANO GRADE: 2016

DATA DA ATIVIDADE	DESCRIÇÃO DA ATIVIDADE	TOTAL DE HORAS	ASSINATURA DO ALUNO	HORAS ATRIBUÍDAS (1)	ASSINATURA DO PROFESSOR
01/10/16	Coleta de dados	03	Felipe Cunha Santos		
02/10/16	"	03	Felipe Cunha Santos		
03/10/16	"	03	Felipe Cunha Santos		
04/10/16	Planejamento	03	Felipe Cunha Santos		
05/10/16	"	03	Felipe Cunha Santos		
06/10/16	"	03	Felipe Cunha Santos		
07/10/16	"	03	Felipe Cunha Santos		
08/10/16	"	03	Felipe Cunha Santos		
09/10/16	"	03	Felipe Cunha Santos		
10/10/16	Alessamento da peça	03	Felipe Cunha Santos		
11/10/16	"	03	Felipe Cunha Santos		
12/10/16	"	03	Felipe Cunha Santos		
13/10/16	"	03	Felipe Cunha Santos		
14/10/16	"	03	Felipe Cunha Santos		
15/10/16	Teste	03	Felipe Cunha Santos		
16/10/16	"	04	Felipe Cunha Santos		

(1) Horas atribuídas de acordo com o regulamento das Atividades Práticas Supervisionadas do curso.

TOTAL DE HORAS ATRIBUÍDAS: 60

AValiação: _____
Aprovado ou Reprovado

NOTA: _____

DATA: / /

CARIMBO E ASSINATURA DO COORDENADOR DO CURSO

Fonte: Própria, 2016.

Imagem 32 – José Vitor Zanoni da Costa.


UNIP
UNIVERSIDADE PAULISTA

FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: Helipe Cunha Santos TURMA: CC3A33 RA: C4695F-2

CURSO: Ciência da Computação CAMPUS: Itatuna SEMESTRE: 3º TURNO: manhã

CÓDIGO DA ATIVIDADE: 385X SEMESTRE: 4º ANO GRADE: 2016

DATA DA ATIVIDADE	DESCRIÇÃO DA ATIVIDADE	TOTAL DE HORAS	ASSINATURA DO ALUNO	HORAS ATRIBUÍDAS (1)	ASSINATURA DO PROFESSOR
01/10/16	Coleta de dados	03	Helipe Cunha Santos		
02/10/16	"	03	Helipe Cunha Santos		
03/10/16	"	03	Helipe Cunha Santos		
04/10/16	Planejamento	03	Helipe Cunha Santos		
05/10/16	"	03	Helipe Cunha Santos		
06/10/16	"	03	Helipe Cunha Santos		
07/10/16	"	03	Helipe Cunha Santos		
08/10/16	"	03	Helipe Cunha Santos		
09/10/16	"	03	Helipe Cunha Santos		
10/10/16	Alessamento da peça	03	Helipe Cunha Santos		
11/10/16	"	03	Helipe Cunha Santos		
12/10/16	"	03	Helipe Cunha Santos		
13/10/16	"	03	Helipe Cunha Santos		
14/10/16	"	03	Helipe Cunha Santos		
15/10/16	Teste	03	Helipe Cunha Santos		
16/10/16	"	04	Helipe Cunha Santos		

(1) Horas atribuídas de acordo com o regulamento das Atividades Práticas Supervisionadas do curso.

TOTAL DE HORAS ATRIBUÍDAS: 60

AValiação: _____
Aprovado ou Reprovado

NOTA: _____

DATA: / / _____

CARIMBO E ASSINATURA DO COORDENADOR DO CURSO

Fonte: Própria, 2016.

ANEXOS

ANEXO A – Explicação de cada API

- *Random*: as instâncias da classe *random* são objetos geradores de números aleatórios, que produzem estes números em resposta a solicitações. Através dela é possível gerar diversos tipos distintos de números aleatórios.
- *ImageIcon*: é uma implementação da interface *Icon* que pinta ícones de imagens. Imagens que são criadas a partir de um URL.
- *Graphics2D*: estende a classe *Graphics* para fornecer um controle melhor sobre geometria, transformações de coordenadas, gerenciamento de cores e layout de texto. Esta é a classe fundamental para renderizar formas bidimensionais, texto e imagens na plataforma Java.
- *Image*: é uma superclasse para todas as classes de imagem que representam imagens gráficas.
- *Rectangle*: essa classe usa inteiros de 32 bits para armazenar a sua localização e as dimensões. Um retângulo especifica uma área em um espaço de coordenadas que é definida pelo ponto superior esquerdo do retângulo (x, y) no espaço de coordenadas, a sua largura, e sua altura. Largura e altura do retângulo são campos públicos.
- *List*: é uma interface que armazena uma “lista” que aceita elementos duplicados.
- *KeyEvent*: um evento que indica que uma tecla foi pressionada ou liberada. Este evento é gerado por um objeto componente (como um campo de texto) quando uma tecla é pressionada. O evento é passado para cada objeto *KeyListener* ou *KeyAdapter* que registrou para receber esses eventos utilizando o método *addKeyListener* do componente.

- *Graphics*: é a classe base abstrata para todos os contextos gráficos que permitem que um aplicativo desenhe em componentes que se realizam em vários dispositivos.
- *ActionEvent*: um evento semântico (relativo ao significado ou ao sentido das unidades linguísticas), que indica que uma ação definida pelo componente ocorreu. Este evento é gerado por um componente (tal como um botão), quando o componente de ação específico ocorre (como ser pressionado), o evento é passado para cada objeto *ActionListener* que o registra para receber esses eventos utilizando o método *addActionListener* do componente.
- *ActionListener*: a interface *Listener* é usada para receber eventos de ação. A classe que está interessada no estudo de um evento de ação implementa essa interface e o objeto criado com essa classe é registrado com um componente, utilizando o método *addActionListener* do componente. Quando o evento de ação ocorre, o método *actionPerformed* desse objeto é invocado.
- *KeyAdapter*: classe abstrata para receber eventos de teclado. Todos os métodos desta classe estão vazios. Esta classe é de conveniência para criar objetos da classe *ActionListener*.
- *JPanel*: é um componente do pacote *swing* que tem como principal função servir de contêiner para outros componentes. Ele vai atuar na parte visual e ajuda na organização dos componentes da interface com o usuário.
- *JInternalFrame*: um objeto que fornece muitas das características de um frame nativo, incluindo arrastar, fechar, ícone, redimensionamento, exibição do título e suporte para uma barra de menu.
- *Time*: aciona um ou mais *ActionEvents* em intervalos especificados. Um exemplo de uso é um objeto de animação que utiliza um temporizador como gatilho para desenhar seus quadros. Configurar um temporizador envolve a criação de um objeto *Timer*, registrar um ou mais *actionListener* sobre ele e iniciar o temporizador usando o método *start*.

- *Color*: é usada para encapsular cores no espaço de cores RGB padrão ou cores em espaços arbitrários identificadas por um *ColorSpace*. Cada cor tem um valor alfa implícito de 1,0 ou uma explícita fornecida no construtor. O valor de alfa define a transparência de uma cor e pode ser representada por um valor fracionários na gama de 0,0 – 1,0 ou 0 – 255. Um valor alfa de 1,0 ou 255 significa que a cor é completamente opaca e um valor 0 ou 0,0 significa que a cor é completamente transparente.
- *Font*: representa fontes de texto que são utilizadas para processar o texto de uma forma visível. Uma fonte fornece as informações necessárias para mapear as sequências de caracteres para sequências de “desenhos” e tornar as sequências de “desenhos” em objetos gráficos e componentes.
- *JOptionPane*: é uma classe que possibilita a criação de uma caixa de diálogo padrão que ou solicita ou valor para o usuário ou retorna uma informação.
- *EventQueue*: é uma classe independente de plataforma que coloca eventos em sequência, tanto das classes de pares subjacente e de classes de aplicações. Existe apenas um *EventQueue* no sistema.
- *ArrayList*: implemente todas as operações de lista opcionais. Além de implementar a interface *List*, essa classe fornece métodos para manipular o tamanho da matriz que é usada internamente para armazenar a lista.
- *ResultSet*: é uma tabela de dados que representa um conjunto de resultados de banco de dados que normalmente é gerado pela execução de uma declaração de consulta ao banco de dados. Um objeto *ResultSet* mantém o cursor apontando para a sua linha de dados atual. Inicialmente o curso é posicionado antes da primeira linha. O próximo método move o cursor para a próxima linha e retorna falso quando não há mais linhas no objeto. Ele pode ser usado em um loop *while* para percorrer o conjunto de resultados. Um objeto *ResultSet* padrão não é atualizável e tem um cursor que se move somente para

frente. Assim, você pode percorrer apenas uma vez, e somente a partir da primeira linha para a última linha.

- *SQLException*: uma exceção que fornece informações sobre um erro de acesso de banco de dados ou outros erros.
- *DefaultListModel*: cria listas imutáveis.
- *EmptyBorder*: uma classe que fornece uma borda vazia transparente que ocupado espaço, mas não tem nenhum “desenho”.
- *Connection*: representa uma conexão com um banco de dados específico. Instruções SQL são executadas e os resultado são devolvidos dentro do contexto de uma ligação.
- *DriverManager*: a classe tentará carregar as classes do driver referenciadas no *jdbc.drivers*, propriedades do sistema. Isso permite que um usuário possa personalizar os drivers JDBC utilizados por seus aplicativos.
- *PreparedStatement*: é um objeto que representa uma instrução SQL pré-compilada. A instrução DQL é pré-compilada e armazenada em um objeto *PreparedStatement*. Este objeto pode ser utilizado de forma eficiente para executar essa declaração várias vezes.
- *FileInputStream*: obtém bytes de entrada a partir de um arquivo em um sistema. Ela é destinada a ler fluxos de bytes crus, como imagens.
- *IOException*: é a classe geral de exceções produzidas por operações de IO com falha ou interrompidas. Essa exceção ocorre quando uma operação IO falhou por algum motivo.
- *AudioPlayer*: é o carro-chefe do pacote *sun.audio*. Ele é usado para reproduzir todos os fluxos que foram criados com as outras classes. Não há construtor na classe. Ele apenas estende da classe *Thread* e fornece os métodos *start* e *stop*.
- *AudioStream*: permite mover para trás e para frente (retroceder e avançar rapidamente) dentro de um arquivo de áudio, além de reproduzir os dados de um áudio do começo ao fim.

ANEXO B – Código da classe Bolha.java

```

package br.com.unip.cc.armas;

import java.util.Random;

import javax.swing.ImageIcon;

import br.com.unip.cc.base.ImageEntity;

/**
 * Esta classe representa a municao gota do personagem e é uma subclasse da ImageEntity
 *
 * @version 0.1 29/Ago/2016
 */

public class Bolha extends ImageEntity {
    private int velocidade = 2;
    private int randomNum = 0;

    /**
     * ao instanciar uma bolha, o cosntrutor busca todas as imagens localizadas na pasta
res
     * recebe dois parametros, do qual instanceara o objeto em X e Y
     * cria uma sequencia de números aleatórios, para que varie as imagens da bolha
     *
     * @param x
     * @param y
     */
    public Bolha(int x, int y) {
        Random rand = new Random();
        ImageIcon referencia1 = new ImageIcon("res\\bolha.png");
        ImageIcon referencia2 = new ImageIcon("res\\bolhas2.png");
        ImageIcon referencia3 = new ImageIcon("res\\bolhas3.png");
        ImageIcon referencia4 = new ImageIcon("res\\bolhas4.png");

        setX(x);
        setY(y);
    }

```



```

        randomNum = rand.nextInt(3);

        if(randomNum == 0)
            setImage(referencia1.getImage());

        else if(randomNum == 1)
            setImage(referencia2.getImage());

        else if(randomNum == 2)
            setImage(referencia3.getImage());

        else if(randomNum == 3)
            setImage(referencia4.getImage());

        else if(randomNum == 4)
            setImage(referencia1.getImage());

        else if(randomNum == 5)
            setImage(referencia2.getImage());

        setVisivel(true);
    }

    public void setVelocidade(int velocidade) {
        this.velocidade = velocidade;
    }

    //Método responsável por movimentar a bolha na tela
    public void mexer() {
        incX(velocidade);

        if(getX() > LARGURA_TELA)
            setVisivel(true);

        else if(getX() < 1)
            setX(1);

        else if(getX() > 1000)
            setX(9000);
    }
}

```

ANEXO C – Código da classe Gota.java

```

package br.com.unip.cc.armas;

import java.util.Random;

import javax.swing.ImageIcon;

import br.com.unip.cc.base.ImageEntity;

/**
 * Esta classe representa a municao gota do personagem e é uma subclasse da ImageEntity
 *
 * @version 0.1 29/Ago/2016
 */

public class Gota extends ImageEntity {
    private int velocidade = 1;
    private int randomNum = 0;

    /**
     * ao instanciar uma gota, o cosntrutor busca todas as imagens localizadas na pasta
res
     * recebe dois parametros, do qual instanceara o objeto em X e Y
     * cria uma sequencia de números aleatórios, para que varie as imagens da gota
     *
     * @param x
     * @param y
     */
    public Gota(int x, int y) {
        Random rand = new Random();
        ImageIcon referencia1 = new ImageIcon("res\\gota.png");
        ImageIcon referencia2 = new ImageIcon("res\\gota2.png");
        ImageIcon referencia3 = new ImageIcon("res\\gota3.png");

        setX(x);
        setY(y);

        randomNum = rand.nextInt(2);
    }

```

```
        if(randomNum == 0)
            setImage(referencia1.getImage());

        else if(randomNum == 1)
            setImage(referencia2.getImage());

        else if(randomNum == 2)
            setImage(referencia3.getImage());

        setVisivel(true);
    }

    public void setVelocidade(int velocidade) {
        this.velocidade = velocidade;
    }

    public void mexer() {
        incX(velocidade);
        incY(velocidade);

        if(getX() > LARGURA_TELA)
            setVisivel(true);

        if(getY() > LARGURA_TELA)
            setVisivel(true);
    }
}
```

ANEXO D – Código da classe BaseGameEntity.java

```
package br.com.unip.cc.base;

/**
 * Esta classe é a base para os demais objetos (inimigos, tiros,..)
 * fundamentais no jogo. Ela serve para movimentar o objeto,
 * modar o angulo do mesmo e verificar se ele está vivo.
 *
 * @version 0.01 29/Ago/2016
 */

public class BaseGameEntity extends Object {
    private boolean alive = false;
    private double x = 0, y = 0;
    private double velX = 0, velY = 0;
    private double moveAngle = 0, faceAngle = 0;

    BaseGameEntity() {
        setAlive(false);
        setX(0.0);
        setY(0.0);
        setVelX(0.0);
        setVelY(0.0);
        setMoveAngle(0.0);
        setFaceAngle(0.0);
    }

    public boolean isAlive() {
        return alive;
    }

    public double getX() {
        return x;
    }

    public double getY() {
        return y;
    }
}
```

```
public double getVelX() {  
    return velX;  
}  
  
public double getVelY() {  
    return velY;  
}  
  
public double getMoveAngle() {  
    return moveAngle;  
}  
  
public double getFaceAngle() {  
    return faceAngle;  
}  
  
public void setAlive(boolean alive) {  
    this.alive = alive;  
}  
  
public void setX(double x) {  
    this.x = x;  
}  
  
public void setY(double y) {  
    this.y = y;  
}  
  
public void setVelX(double x) {  
    this.velX = x;  
}  
  
public void setVelY(double velY) {  
    this.velY = velY;  
}  
  
public void setFaceAngle(double angle) {  
    this.faceAngle = angle;  
}
```

```
public void setMoveAngle(double angle) {
    this.moveAngle = angle;
}

/**Métodos de incremento*/
public void incX(double i) {
    this.x += i;
}

public void incY(double i) {
    this.y += i;
}

public void incVelX(double i) {
    this.velX += i;
}

public void incVelY(double i) {
    this.velY += i;
}

public void incFaceAngle(double i) {
    this.faceAngle += i;
}
public void incMoveAngle(double i) {
    this.moveAngle += i;
}
}
```

ANEXO E – Código da classe ImageEntity.java

```
package br.com.unip.cc.base;

import java.awt.Graphics2D;
import java.awt.Image;
import java.awt.Rectangle;

/**
 * Esta classe possibilita a utilização de uma imagem para os objetos no game
 *
 * @version 0.20 31/Jul/2016
 */

public class ImageEntity extends BaseGameEntity {
    protected Image image = null; // Objeto que irá armazenar a imagem
    private Graphics2D g2d = null; // Objeto que controla a geometria, transformações de
    coordenadas, gerenciamento de cores e layout de texto
    protected boolean visivel = false;
    protected static final int LARGURA_TELA = 997;

    public ImageEntity() {
        setImage(null);
        setAlive(true);
    }

    public Image getImage() {
        return image;
    }

    public void setImage(Image image) {
        this.image = image;
    }

    public void setGraphics(Graphics2D g) {
        g2d = g;
    }

    public boolean isVisivel() {
```

```
        return visivel;
    }

    public void setVisivel(boolean visivel) {
        this.visivel = visivel;
    }

    public int getWidth() {
        return image.getWidth(null);
    }

    public int getHeight() {
        return image.getHeight(null);
    }

    //Método que irá desenhar a imagem na tela
    public void draw(int x, int y) {
        g2d.drawImage(getImage(), x, y, null);
    }

    //Método que cuida da máscara de cada imagem
    public Rectangle getBounds() {
        return new Rectangle((int)getX(), (int)getY(), getWidth(), getHeight());
    }
}
```


ANEXO F – Código da classe Personagem.java

```
package br.com.unip.cc.base;

import java.awt.Rectangle;
import java.awt.event.KeyEvent;
import java.util.List;

import br.com.unip.cc.armas.Bolha;
import br.com.unip.cc.armas.Gota;

/**
 * esta interface é usada para fazer uma conversao ampliada
 */

public interface Personagem {
    void mexer();
    List<Bolha> getBolhas();
    List<Gota> getGotas();
    void atira();
    void atira2();
    void keyPressed(KeyEvent tecla);
    void keyReleased(KeyEvent tecla);
    int getVida();
    void setVida(int vida);
    void incVida(int vida);
    void decVida(int vida);
    void setVisivel(boolean b);
    Rectangle getBounds();
}
```

ANEXO G – Código da classe Batalha.java

```

package br.com.unip.cc.control;

import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Image;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.util.List;

import javax.swing.ImageIcon;
import javax.swing.JInternalFrame;
import javax.swing.JPanel;
import javax.swing.Timer;

import br.com.unip.cc.armas.Bolha;
import br.com.unip.cc.armas.Gota;
import br.com.unip.cc.personagem.PersonagemBlue;
import br.com.unip.cc.personagem.PersonagemRed;

/**
 * Esta classe controla os eventos da batalha
 *
 * @version 0.1 09/Set/2016
 */

@SuppressWarnings("serial")
public class Batalha extends JPanel implements ActionListener{
    private final ImageIcon REFERENCIA = new ImageIcon("res\\fundojogo0.jpg");
    private final Image fundo = REFERENCIA.getImage();
    private Timer timer = null;
    private PersonagemBlue personagemBlue = null;
    private PersonagemRed personagemRed = null;
    private List<Bolha> bolhasBlue = null;
    private List<Gota> gotasBlue = null;
    private List<Bolha> bolhasRed = null;

```

```

private List<Gota> gotasRed = null;

public Batalha() {
    setFocusable(true);
    setDoubleBuffered(true);
    addKeyListener(new TecladoAdapter());

    personagemRed = new PersonagemRed(true);
    personagemBlue = new PersonagemBlue(true);
    timer = new Timer(1, this);

    setLayout(null);

    JInternalFrame internalFrame = new JInternalFrame("Batalha");
    internalFrame.setBounds(0, 0, 450, 300);
    add(internalFrame);
    internalFrame.setVisible(true);
    timer.start();
}

//Método que serve para desenhar no JPanel
public void paint(Graphics g) {
    Graphics2D graficos = (Graphics2D) g;
    graficos.drawImage(fundo, 0, 0, null);
    graficos.drawImage(personagemBlue.getImage(), (int)
personagemBlue.getX(), (int) personagemBlue.getY(), this);
    graficos.drawImage(personagemRed.getImage(), (int)
personagemRed.getX(), (int) personagemRed.getY(), this);

    bolhasBlue = personagemBlue.getBolhas();
    gotasBlue = personagemBlue.getGotas();
    bolhasRed = personagemRed.getBolhas();
    gotasRed = personagemRed.getGotas();

    for (int i = 0; i < gotasBlue.size(); i++) {
        Gota n = (Gota) gotasBlue.get(i);
        graficos.drawImage(n.getImage(), (int) n.getX(), (int) n.getY(), this);
    }

    for (int i = 0; i < bolhasBlue.size(); i++) {

```

```

        Bolha m = (Bolha) bolhasBlue.get(i);
        graficos.drawImage(m.getImage(), (int) m.getX(), (int) m.getY(), this);
    }

    for (int i = 0; i < gotasRed.size(); i++) {
        Gota n = (Gota) gotasRed.get(i);
        graficos.drawImage(n.getImage(), (int) n.getX(), (int) n.getY(), this);
    }

    for (int i = 0; i < bolhasRed.size(); i++) {
        Bolha m = (Bolha) bolhasRed.get(i);
        graficos.drawImage(m.getImage(), (int) m.getX(), (int) m.getY(), this);
    }

    if (personagemBlue.getVida() == 0) {
        ImageIcon fimJogo = new ImageIcon("res\\Amoeba_Venceu.png");
        graficos.drawImage(fimJogo.getImage(), 0, 0, null);
    }

    if (personagemRed.getVida() == 0) {
        ImageIcon fimJogo = new ImageIcon("res\\Ameba_Venceu.png");
        graficos.drawImage(fimJogo.getImage(), 0, 0, null);
    }
}

//verifica e executa todas as ações que ocorrem dentro do JPanel
public void actionPerformed(ActionEvent arg0) {
    bolhasBlue = personagemBlue.getBolhas();
    gotasBlue = personagemBlue.getGotas();
    bolhasRed = personagemRed.getBolhas();
    gotasRed = personagemRed.getGotas();

    for (int i = 0; i < bolhasBlue.size(); i++) {
        Bolha b = (Bolha) bolhasBlue.get(i);

        if (b.isVsivel()) {
            b.mexer();
            if (personagemBlue.getX() >= 465)
                bolhasBlue.get(i).setVelocidade(-2);
        }
    }
}

```

```

else
    bolhasBlue.remove(i);

    if(bolhasBlue.get(i).getX() > 960 || bolhasBlue.get(i).getX() < 2)
        bolhasBlue.remove(i);
}

for (int i = 0; i < gotasBlue.size(); i++) {
    Gota g = (Gota) gotasBlue.get(i);

    if (g.isVsivel()){
        g.mexer();
        if(personagemBlue.getX() >= 465)
            gotasBlue.get(i).incX(-2);
    }

    else
        gotasBlue.remove(i);

    if(gotasBlue.get(i).getY() > 672)
        gotasBlue.remove(i);
}

for (int i = 0; i < bolhasRed.size(); i++) {
    Bolha b = (Bolha) bolhasRed.get(i);

    if (b.isVsivel()) {
        b.mexer();
        if(personagemRed.getX() <= 465)
            bolhasRed.get(i).setVelocidade(2);

        else
            bolhasRed.get(i).setVelocidade(-2);
    }

    else
        bolhasRed.remove(i);

    if(bolhasRed.get(i).getX() > 960 || bolhasRed.get(i).getX() < 2)

```

```

        bolhasRed.remove(i);
    }

    for (int i = 0; i < gotasRed.size(); i++) {
        Gota g = (Gota) gotasRed.get(i);

        if (g.isVsivel()) {
            g.mexer();
            if(personagemRed.getX() >= 465)
                gotasRed.get(i).incX(-2);
        }

        else
            gotasRed.remove(i);

        if(gotasRed.get(i).getY() > 672 || gotasRed.get(i).getY() == 0)
            gotasRed.remove(i);
    }

    colisao();
    personagemBlue.mexer();
    personagemRed.mexer();
    repaint();
}

public void colisao() {
    bolhasBlue = personagemBlue.getBolhas();
    gotasBlue = personagemBlue.getGotas();
    bolhasRed = personagemRed.getBolhas();
    gotasRed = personagemRed.getGotas();

    for (int i = 0; i < bolhasBlue.size(); i++)
        if
(personagemRed.getBounds().intersects(bolhasBlue.get(i).getBounds())) {
            bolhasBlue.remove(i);
            personagemRed.decVida(1);
        }

    for (int i = 0; i < gotasBlue.size(); i++)

```

```

        if
(personagemRed.getBounds().intersects(gotasBlue.get(i).getBounds())) {
            gotasBlue.remove(i);
            personagemRed.decVida(1);
        }

        for (int i = 0; i < bolhasRed.size(); i++)
            if
(personagemBlue.getBounds().intersects(bolhasRed.get(i).getBounds())) {
                bolhasRed.remove(i);
                personagemBlue.decVida(1);
            }

        for (int i = 0; i < gotasRed.size(); i++)
            if
(personagemBlue.getBounds().intersects(gotasRed.get(i).getBounds())) {
                gotasRed.remove(i);
                personagemBlue.decVida(1);
            }
    }

    //Caso o usuário aperte enter no menu inicial
    private class TecladoAdapter extends KeyAdapter {
        public void keyPressed(KeyEvent e) {
            personagemBlue.keyPressed(e);
            personagemRed.keyPressed(e);
        }

        public void keyReleased(KeyEvent e) {
            personagemBlue.keyReleased(e);
            personagemRed.keyReleased(e);
        }
    }
}

```

ANEXO H – Código da classe Control.java

```
package br.com.unip.cc.control;

/**
 * Esta classe controla os pontos
 *
 * @version 0.1 09/Set/2016
 */

public class Control {
    private static int pontos = 0;

    public Control() {
        pontos = 0;
    }

    public static void setPontos(int pontos) {
        Control.pontos = pontos;
    }

    public static void incPontos(int pontos) {
        Control.pontos += pontos;
    }

    public static int getPontos() {
        return pontos;
    }
}
```


ANEXO I – Código da classe Fase.java

```
package br.com.unip.cc.control;

import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.Image;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.KeyAdapter;
import java.awt.event.KeyEvent;
import java.util.Calendar;
import java.util.GregorianCalendar;
import java.util.List;

import javax.persistence.EntityManager;
import javax.swing.ImageIcon;
import javax.swing.Timer;
import javax.swing.JInternalFrame;
import javax.swing.JOptionPane;
import javax.swing.JPanel;

import br.com.unip.cc.armas.Bolha;
import br.com.unip.cc.armas.Gota;
import br.com.unip.cc.dao.JogadorDAO;
import br.com.unip.cc.dao.RecordeDAO;
import br.com.unip.cc.domain.Jogadores;
import br.com.unip.cc.domain.Recordes;
import br.com.unip.cc.fases.Fase01;
import br.com.unip.cc.fases.Fase02;
import br.com.unip.cc.fases.Fase03;
import br.com.unip.cc.fases.Fase04;
import br.com.unip.cc.fases.Fase05;
import br.com.unip.cc.inimigos.Boss;
//import br.com.unip.cc.jdbc.Jogador;
//import br.com.unip.cc.jdbc.JogadorDAO;
//import br.com.unip.cc.jdbc.RecordeDAO;
```

```

import br.com.unip.cc.persistence.EntityManagerProvider;
import br.com.unip.cc.personagem.PersonagemBlue;

/**
 * Esta classe representa o controle das fases. Ela verifica se o jogador pode continuar
 * se é fim de jogo, invoca os métodos de cada fase para verificar a colisão, iniciar
 * os inimigos e etc..
 * É uma subclasse de JPanel com a implementação da ActionListener
 *
 * @version 0.1 02/Set/2016
 */

@SuppressWarnings("serial")
public class Fase extends JPanel implements ActionListener {
    private Image fundo = null;
    private Timer timer = null;
    private PersonagemBlue personagem = null;
    private Fase01 fase01 = null;
    private Fase02 fase02 = null;
    private Fase03 fase03 = null;
    private Fase04 fase04 = null;
    private Fase05 fase05 = null;
    private List<Bolha> bolhas = null;
    private List<Gota> gotas = null;
    private boolean emJogo = false, emJogo2 = false;
    private int cont = 0;
    public Fase() {
        setFocusable(true);
        setDoubleBuffered(true);
        addKeyListener(new TecladoAdapter());

        fase01 = new Fase01();
        fase02 = new Fase02();
        fase03 = new Fase03();
        fase04 = new Fase04();
        fase05 = new Fase05();
        personagem = new PersonagemBlue(false);
        timer = new Timer(1, this);

        emJogo2 = true;
    }

```

```

        setLayout(null);

        JInternalFrame internalFrame = new JInternalFrame("Fases");
        internalFrame.setBounds(0, 0, 450, 300);
        add(internalFrame);
        internalFrame.setVisible(true);
        timer.start();
    }

    //Método que serve para desenhar no JPanel
    public void paint(Graphics g) {
        Graphics2D graficos = (Graphics2D) g;
        graficos.drawImage(fundo, 0, 0, null);

        if (emJogo) {
            bolhas = personagem.getBolhas();
            gotas = personagem.getGotas();

            graficos.drawImage(personagem.getImage(), (int) personagem.getX(),
            (int) personagem.getY(), this);

            for (int i = 0; i < personagem.sizeGotas(); i++)
                graficos.drawImage(personagem.getGotas(i).getImage(), (int)
            personagem.getGotas(i).getX(), (int) personagem.getGotas(i).getY(), this);

            for (int i = 0; i < personagem.sizeBolhas(); i++)
                graficos.drawImage(personagem.getBolhas(i).getImage(), (int)
            personagem.getBolhas(i).getX(), (int) personagem.getBolhas(i).getY(), this);

            if(fase01.isAtiva())
                for (int i = 0; i < fase01.sizeInimigo(); i++)
                    graficos.drawImage(fase01.getInimigo(i).getImage(),
            (int) fase01.getInimigo(i).getX(), (int) fase01.getInimigo(i).getY(), this);

            if(fase02.isAtiva())
                for (int i = 0; i < fase02.sizeInimigo(); i++)
                    graficos.drawImage(fase02.getInimigo(i).getImage(),
            (int) fase02.getInimigo(i).getX(), (int) fase02.getInimigo(i).getY(), this);

```

```

        if(fase03.isAtiva())
            for (int i = 0; i < fase03.sizeInimigo(); i++)
                graficos.drawImage(fase03.getInimigo(i).getImage(),
(int) fase03.getInimigo(i).getX(), (int) fase03.getInimigo(i).getY(), this);

        if(fase04.isAtiva())
            for (int i = 0; i < fase04.sizeInimigo(); i++)
                graficos.drawImage(fase04.getInimigo(i).getImage(),
(int) fase04.getInimigo(i).getX(), (int) fase04.getInimigo(i).getY(), this);

        if(fase05.isAtiva())
            for (int i = 0; i < fase05.sizeInimigo(); i++)
                graficos.drawImage(fase05.getInimigo(i).getImage(),
(int) fase05.getInimigo(i).getX(), (int) fase05.getInimigo(i).getY(), this);

        if (this.cont == 4) {
            Font font = new Font("Serif", Font.PLAIN, 20);
            graficos.setFont(font);
            graficos.setColor(Color.white);
            graficos.drawString("Vida do Boss: " + Boss.getVida(), 445,
35);
        }

        if(fase05.isAtiva())
            if (fase05.sizeInimigo() == 0) {
                Imagemcon fimJogo = new
Imagemcon("res\\fim_jogo.png");

                graficos.drawImage(fimJogo.getImage(), 0, 0, null);
                g.dispose();
            }
        }

        if (emJogo2 == true) {
            Imagemcon fimJogo = new Imagemcon("res\\tuto1.png");
            graficos.drawImage(fimJogo.getImage(), 0, 0, null);
            g.dispose();
        }

```

```

else if (emJogo == false) {
    ImageIcon fimJogo = new ImageIcon("res\\game_over.png");
    graficos.drawImage(fimJogo.getImage(), 0, 0, null);
}
}

//verifica e executa todas as ações que ocorrem dentro do JPanel
public void actionPerformed(ActionEvent arg0) {
    bolhas = personagem.getBolhas();
    gotas = personagem.getGotas();

    if(emJogo) {
        for (int i = 0; i < bolhas.size(); i++) {
            if (bolhas.get(i).isVsivel()) {
                bolhas.get(i).mexer();

                if(personagem.getX() >= 465)
                    bolhas.get(i).setVelocidade(-2);
            }

            else

                bolhas.remove(i);

            if(bolhas.get(i).getX() > 960 || bolhas.get(i).getX() < 2)
                bolhas.remove(i);
        }

        for (int i = 0; i < gotas.size(); i++) {
            if (gotas.get(i).isVsivel()) {
                gotas.get(i).mexer();

                if(personagem.getX() >= 465)
                    gotas.get(i).incX(-2);
            }

            else

                gotas.remove(i);

            if(gotas.get(i).getY() > 672)
                gotas.remove(i);
        }
    }
}

```

```

}

if(fase01.isAtiva()) {
    for (int i = 0; i < fase01.sizeInimigo(); i++) {
        if (fase01.getInimigo(i).isVsivel())
            fase01.getInimigo(i).mexer();
        else
            fase01.deleteInimigo(i);
    }

    if (fase01.sizeInimigo() == 0 && this.cont == 0) {
        this.cont++;
        fase01.setAtiva(false);
        fundo = fase02.getReferencia().getImage();
        fase02.inicializaInimigo2();
    }
}

if(fase02.isAtiva()){
    for (int e = 0; e < fase02.sizeInimigo(); e++) {
        if (fase02.getInimigo(e).isVsivel())
            fase02.getInimigo(e).mexer();
        else
            fase02.deleteInimigo(e);
    }

    if (fase02.sizeInimigo() == 0 && this.cont == 1) {
        this.cont++;
        fase02.setAtiva(false);
        fundo = fase03.getReferencia().getImage();
        fase03.inicializaInimigo3();
    }
}

if(fase03.isAtiva()) {
    for (int b = 0; b < fase03.sizeInimigo(); b++) {
        if (fase03.getInimigo(b).isVsivel())
            fase03.getInimigo(b).mexer();
        else
            fase03.deleteInimigo(b);
    }
}

```

```

    }

    if (fase03.sizeInimigo() == 0 && this.cont == 2) {
        this.cont++;
        fase03.setAtiva(false);
        fundo = fase04.getReferencia().getImage();
        fase04.inicializaInimigo4();
    }
}

if(fase04.isAtiva()) {
    for (int c = 0; c < fase04.sizeInimigo(); c++) {
        if (fase04.getInimigo(c).isVsivel())
            fase04.getInimigo(c).mexer();
        else
            fase04.deleteInimigo(c);
    }

    if (fase04.sizeInimigo() == 0 && this.cont == 3) {
        this.cont++;
        fase04.setAtiva(false);
        fundo = fase05.getReferencia().getImage();
        fase05.inicializaBoss();
    }
}

if(fase05.isAtiva())
    for (int p = 0; p < fase05.sizeInimigo(); p++) {
        fase05.getInimigo(p).mexer();

        if (fase05.getInimigo(p).isVsivel()) {
            fase05.getInimigo(0);
            fase05.getInimigo(p).mexer();
        }

        else if (Boss.getVida() == 0)
            fase05.deleteInimigo(p);
    }
personagem.mexer();
colisao();

```

```

    }
    repaint();
}

//verificando as colisões
public void colisao() {
    if(fase01.isAtiva())
        if(fase01.isColisao(personagem, bolhas, gotas)){
            if(personagem.getVida() == 0)
                cadastre();
            emJogo = false;
        }

    if(fase02.isAtiva())
        if(fase02.isColisao(personagem, bolhas, gotas)){
            if(personagem.getVida() == 0)
                cadastre();
            emJogo = false;
        }

    if(fase03.isAtiva())
        if(fase03.isColisao(personagem, bolhas, gotas)){
            if(personagem.getVida() == 0)
                cadastre();
            emJogo = false;
        }

    if(fase04.isAtiva())
        if(fase04.isColisao(personagem, bolhas, gotas)){
            if(personagem.getVida() == 0)
                cadastre();
            emJogo = false;
        }

    if(fase05.isAtiva())
        if(fase05.isColisao(personagem, bolhas, gotas)){
            if(personagem.getVida() == 0)
                cadastre();
            emJogo = false;
        }
}

```



```

    }

    public void cadastre() {
        /*Jogador player = new Jogador();
        JogadorDAO salve = new JogadorDAO();
        RecordeDAO record = new RecordeDAO();

        player.setNome(JOptionPane.showInputDialog(null, "Digite seu nome",
"Nome", JOptionPane.PLAIN_MESSAGE));
        player.setNick(JOptionPane.showInputDialog(null, "Digite seu apelido",
"Apelido", JOptionPane.PLAIN_MESSAGE));
        player.setRecord(Control.getPontos());
        record.inserirRecorde(player);
        JOptionPane.showMessageDialog(null, salve.inserirPlayer(player));*/

        String nome = JOptionPane.showInputDialog(null, "Digite seu nome", "Nome",
JOptionPane.PLAIN_MESSAGE);

        if(nome != null) {
            Jogadores player = new Jogadores();

            player.setJog_nome(nome);

            nome = null;
            nome = JOptionPane.showInputDialog(null, "Digite seu apelido",
"Apelido", JOptionPane.PLAIN_MESSAGE);

            if(nome != null) {
                Calendar data = GregorianCalendar.getInstance();
                EntityManager em =
EntityManagerProvider.getEntityManagerInstance();

                JogadorDAO daoJogador = new JogadorDAO(em);

                Recordes pontos = new Recordes();
                RecordeDAO daoRecorde = new RecordeDAO(em);

                player.setJog_apelido(nome.substring(0, 4));
                player.setJog_dt_criacao(data.getTime());
                daoJogador.save(player);
            }
        }
    }
}

```

```

        pontos.setRec_quantidade(Control.getPontos());
        pontos.setRec_dt_criacao(data.getTime());
        daoRecorde.save(pontos);
    }
}

//Caso o usuário aperte enter no menu inicial
private class TecladoAdapter extends KeyAdapter {
    public void keyPressed(KeyEvent e) {
        if(e.getKeyCode() == KeyEvent.VK_ENTER){
            emJogo = true;
            emJogo2 = false;
            fundo = fase01.getReferencia().getImage();
            personagem = new PersonagemBlue(false);

            fase01.inicializaInimigo1();
            fase02.setAtiva(false);
            fase03.setAtiva(false);
            fase04.setAtiva(false);
            fase05.setAtiva(false);
            cont = 0;
        }

        personagem.keyPressed(e);
    }

    public void keyReleased(KeyEvent e) {
        personagem.keyReleased(e);
    }
}

```

ANEXO J – Código da classePrincipal.java

```
package br.com.unip.cc.control;

import java.awt.EventQueue;

import br.com.unip.cc.janelas.JFHome;

/**
 * Esta classe inicia o jogo
 *
 * @version 0.1 09/Set/2016
 */

public class Principal {
    public static void main(String[] args) {
        EventQueue.invokeLater(new Runnable() {
            @SuppressWarnings("unused")
            public void run() {
                try {
                    JFHome home = new JFHome();
                } catch (Exception e) {
                    e.printStackTrace();
                }
            }
        });
    }
}
```

ANEXO K – Código da classe GenericDao.java

```
package br.com.unip.cc.dao;

import java.util.List;

import javax.persistence.EntityManager;
import javax.persistence.EntityTransaction;

/**
 * Classe que facilita a comunicacao entre as camadas de negocios
 *
 * @version 0.1 16/Out/2016
 */

public abstract class GenericDao<T> {

    protected EntityManager em;
    Class<T> clazz;

    public GenericDao(EntityManager em, Class<T> clazz) {
        this.em = em;
        this.clazz = clazz;
    }

    public void save(T entity) {
        EntityTransaction trans = em.getTransaction();

        trans.begin();
        try {
            em.persist(entity);
            trans.commit();
        } catch (Exception e) {
            trans.rollback();
        }
    }

    public void delete(T entity) {
```

```
        EntityTransaction trans = em.getTransaction();

        trans.begin();
        try {
            em.remove(entity);
            trans.commit();
        } catch (Exception e) {
            trans.rollback();
        }
    }

    public T findByld(Long id) {
        return em.find(clazz, id);
    }

    @SuppressWarnings("unchecked")
    public List<T> list() {
        return em.createQuery("from " + clazz.getName()).getResultList();
    }
}
```

ANEXO L – Código da classe JogadorDAO.java

```
package br.com.unip.cc.dao;

import javax.persistence.EntityManager;

import br.com.unip.cc.domain.Jogadores;

/**
 * Classe que facilita a comunicacao entre as camadas de negocios
 *
 * @version 0.1 16/Out/2016
 */

public class JogadorDAO extends GenericDao<Jogadores> {
    public JogadorDAO(EntityManager em) {
        super(em, Jogadores.class);
    }
}
```

ANEXO M – Código da classe RecordeDAO.java

```
package br.com.unip.cc.dao;

import javax.persistence.EntityManager;

import br.com.unip.cc.domain.Recordes;

/**
 * Classe que facilita a comunicacao entre as camadas de negocios
 *
 * @version 0.1 16/Out/2016
 */

public class RecordeDAO extends GenericDao<Recordes>{
    public RecordeDAO(EntityManager em) {
        super(em, Recordes.class);
    }
}
```

ANEXO N – Código da classe Jogadores.java

```

package br.com.unip.cc.domain;

import java.util.Date;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.SequenceGenerator;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;

/**
 * Classe que representa a tabela jogadores do banco
 *
 * @version 0.1 16/Out/2016
 */

@Entity //indica para o Hibernate que a classe Jogadores deve ser armazenada em uma
tablea
@Table(name = "jogadores") //Passa o nome da tabela para o Hibernate configurar

public class Jogadores {

    @Id //Configura um atributo como sendo a chave primária da tabela
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "INC_JOG")
    //definição de como será tratada a PK
    @SequenceGenerator(name = "INC_JOG", sequenceName = "INC_JOG",
allocationSize = 1, initialValue = 0)
    @Column(name = "jog_codigo", nullable = false)
    private int jog_codigo;

    @Column(name = "jog_nome", length = 30, nullable = false) //Indica uma coluna de
tamanho 30 e não nulas
    private String jog_nome;

```



```
@Column(name = "jog_apelido", length = 04, nullable = false)
private String jog_apelido;

@Temporal(TemporalType.DATE) //Indica um dado do tipo data
@Column(name = "jog_dt_criacao", nullable = false)
private Date jog_dt_criacao;

public Jogadores() {
}

public int getJog_codigo() {
    return jog_codigo;
}

public void setJog_codigo(int jog_codigo) {
    this.jog_codigo = jog_codigo;
}

public String getJog_nome() {
    return jog_nome;
}

public void setJog_nome(String jog_nome) {
    this.jog_nome = jog_nome;
}

public String getJog_apelido() {
    return jog_apelido;
}

public void setJog_apelido(String jog_apelido) {
    this.jog_apelido = jog_apelido;
}

public Date getJog_dt_criacao() {
    return jog_dt_criacao;
}

public void setJog_dt_criacao(Date jog_dt_criacao) {
```

```
        this.jog_dt_criacao = jog_dt_criacao;
    }
}
```

ANEXO O – Código da classe Recordes.java

```

package br.com.unip.cc.domain;

import java.util.Date;

import javax.persistence.Column;
import javax.persistence.Entity;
import javax.persistence.GeneratedValue;
import javax.persistence.GenerationType;
import javax.persistence.Id;
import javax.persistence.SequenceGenerator;
import javax.persistence.Table;
import javax.persistence.Temporal;
import javax.persistence.TemporalType;

/**
 * Classe que representa a tabela recordes do banco
 *
 * @version 0.1 16/Out/2016
 */

@Entity //indica para o Hibernate que a classe Jogadores deve ser armazenada em uma
tablea
@Table(name = "recordes") //Passa o nome da tabela para o Hibernate configurar

public class Recordes {

    @Id //Configura um atributo como sendo a chave primária da tabela
    @GeneratedValue(strategy = GenerationType.SEQUENCE, generator = "INC_REC")
//definição de como será tratada a PK
    @SequenceGenerator(name = "INC_REC", sequenceName = "INC_REC",
allocationSize = 1, initialValue = 0)
    @Column(name = "rec_codigo", nullable = false)
    private int rec_codigo;

    @Column(name = "rec_quantidade", nullable = false) //Indica uma coluna
    private int rec_quantidade;

```

```
@Temporal(TemporalType.DATE) //Indica um dado do tipo data
@Column(name = "rec_dt_criacao", nullable = false)
private Date rec_dt_criacao;

public Recordes() {
}

public int getRec_codigo() {
    return rec_codigo;
}

public void setRec_codigo(int rec_codigo) {
    this.rec_codigo = rec_codigo;
}

public int getRec_quantidade() {
    return rec_quantidade;
}

public void setRec_quantidade(int rec_quantidade) {
    this.rec_quantidade = rec_quantidade;
}

public Date getRec_dt_criacao() {
    return rec_dt_criacao;
}

public void setRec_dt_criacao(Date rec_dt_criacao) {
    this.rec_dt_criacao = rec_dt_criacao;
}
}
```

ANEXO P – Código da classeFase01.java

```

package br.com.unip.cc.fases;

import java.util.ArrayList;
import java.util.List;

import javax.swing.ImageIcon;

import br.com.unip.cc.armas.Bolha;
import br.com.unip.cc.armas.Gota;
import br.com.unip.cc.base.Personagem;
import br.com.unip.cc.control.Control;
import br.com.unip.cc.inimigos.Inimigo1;
import br.com.unip.cc.personagem.PersonagemBlue;
import br.com.unip.cc.personagem.PersonagemRed;

/**
 * Esta classe manipula os eventos da fase01
 *
 * @version 0.1 29/Ago/2016
 */

public class Fase01 {
    private final ImageIcon REFERENCIA = new ImageIcon("res\\fundejogo1.png");
//pegando a imagem de fundo
    private List<Inimigo1> inimigo = null; //uma lista de objetos do tipo Inimigo1
    private Personagem personagem = null;
    public boolean ativa = false;
    private int[][] coordenadas = {
        {2374, 250}, {2655, 150}, {6445, 200}, {4537, 363},
        {8999, 550}, {4444, 300}, {3671, 450}, {5447, 300}, {7500, 200},
        {6784, 450}, {3576, 210}, {7782, 420}, {6674, 490}, {9788, 150},
        {10011, 500}, {11787, 320}, {9990, 145}, {8889, 200}, {3010, 490},
        {2788, 530}, {5555, 490}, {2580, 550}, {8874, 050}, {15783, 100},
        {4457, 160}, {7777, 050}, {6666, 020}, {2545, 045}, {6119, 065},
        {9957, 460}, {2222, 250}, {2420, 390}, {8952, 430}, {7991, 410},
        {3005, 420}, {2209, 490}, {7030, 350}, {3020, 300}, {11547, 320},
        {20500, 345}, {18777, 300}, {15789, 490}, {11111, 530}, {13547, 490},

```

```

{3578, 550}, {9991, 80}, {6557, 190}, {2333, 360}, {3190, 040},
{10002, 050}, {9898, 445}, {3211, 025}, {2358, 160}, {9124, 350},
{2998, 390}, {3858, 330}, {21111, 210}, {5578, 420}, {9135, 490},
{10123, 150}, {2786, 500}, {6547, 320}, {7778, 145}, {2587, 200},
{21589, 490}, {9805, 530}, {11190, 490}, {7055, 550}, {13559, 050}
};

```

```

public Imagem getReferencia() {
    return this.REFERENCIA;
}

```

```

public Inimigo1 getInimigo(int i) {
    return this.inimigo.get(i);
}

```

```

public int sizeInimigo() {
    return this.inimigo.size();
}

```

```

public void deleteInimigo(int i) {
    this.inimigo.remove(i);
}

```

```

public void inicializaInimigo1() {
    this.inimigo = new ArrayList<Inimigo1>(); //objeto que aponta para uma lista
de inimigos

```

```

    for (int i = 0; i < this.coordenadas.length; i++) //adicionando inimigos a lista
        this.inimigo.add(new Inimigo1(this.coordenadas[i][0],
this.coordenadas[i][1]));
    this.ativa = true;
}

```

```

public void setAtiva(boolean ativa) {
    this.ativa = ativa;
    this.inimigo = null;
}

```

```

public boolean isAtiva() {
    return this.ativa;
}

```

```

    }

    //método que cuida das colisões da fase 01
    public boolean isColisao(Personagem tipo, List<Bolha> bolhas, List<Gota> gotas) {
        if(tipo instanceof PersonagemBlue) {
            this.personagem = (PersonagemBlue) tipo;
        }

        else if(tipo instanceof PersonagemRed){
            this.personagem = (PersonagemRed) tipo;
        }

        bolhas = this.personagem.getBolhas();
        gotas = this.personagem.getGotas();

        for (int i = 0; i < sizeInimigo(); i++)
            if (this.personagem.getBounds().intersects(getInimigo(i).getBounds()))
            {

                this.personagem.setVisivel(false);
                deleteInimigo(i);
                this.personagem.decVida(1);

                if(this.personagem.getVida() == 0)
                    return true;
            }

        for (int j = 0; j < bolhas.size() - 1; j++)
            for (int k = 0; k < sizeInimigo(); k++) {
                if
                (bolhas.get(j).getBounds().intersects(getInimigo(k).getBounds())) {
                    deleteInimigo(k);
                    bolhas.remove(j);
                    Control.incPontos(1);
                }

                else if(bolhas.get(j).getX() > 960)
                    bolhas.remove(j);
            }

        for (int k = 0; k < gotas.size() - 1; k++)

```

```
        for (int l = 0; l < sizeInimigo(); l++) {  
            if  
(gotas.get(k).getBounds().intersects(getInimigo(l).getBounds())) {  
                deleteInimigo(l);  
                gotas.remove(k);  
                Control.incPontos(1);  
            }  
  
            else if(gotas.get(k).getY() > 672)  
                gotas.remove(k);  
        }  
  
        return false;  
    }  
}
```


ANEXO Q – Código da classeFase02.java

```

package br.com.unip.cc.fases;

import java.util.ArrayList;
import java.util.List;

import javax.swing.ImageIcon;

import br.com.unip.cc.armas.Bolha;
import br.com.unip.cc.armas.Gota;
import br.com.unip.cc.base.Personagem;
import br.com.unip.cc.control.Control;
import br.com.unip.cc.inimigos.Inimigo2;
import br.com.unip.cc.personagem.PersonagemBlue;
import br.com.unip.cc.personagem.PersonagemRed;

/**
 * Esta classe manipula os eventos da fase02
 *
 * @version 0.1 29/Ago/2016
 */

public class Fase02 {
    private final ImageIcon REFERENCIA = new ImageIcon("res\\fundojogo2.png");
//pegando a imagem de fundo
    private List<Inimigo2> inimigo = null; //uma lista de objetos do tipo Inimigo2
    private Personagem personagem = null;
    private boolean ativa = false;
    private int[][] coordenadas = {
        {2200, 050}, {2752, 150}, {3057, 200}, {4856, 250},
        {13333, 350}, {9546, 450}, {7854, 530}, {4688, 130},
        {5113, 060}, {9754, 290}, {21333, 050}, {9785, 150},
        {2198, 330}, {3511, 430}, {4669, 500}, {3020, 400},
        {7888, 200}, {8455, 250}, {6643, 450}, {5555, 450},
        {2222, 530}, {6464, 90}, {2256, 500}, {3333, 400},
        {12664, 230}, {20000, 290}, {4587, 70}, {12120, 40}
    };

```

```

public Imagem getReferencia() {
    return this.REFERENCIA;
}

public Inimigo2 getInimigo(int i) {
    return this.inimigo.get(i);
}

public int sizeInimigo() {
    return this.inimigo.size();
}

public void deleteInimigo(int i) {
    this.inimigo.remove(i);
}

public void inicializaInimigo2() {
    this.inimigo = new ArrayList<Inimigo2>();//objeto que aponta para uma lista de
inimigos

    for (int i = 0; i < this.coordenadas.length; i++) //adicionando inimigos a lista
        this.inimigo.add(new Inimigo2(this.coordenadas[i][0],
this.coordenadas[i][1]));
    this.ativa = true;
}

public void setAtiva(boolean ativa) {
    this.ativa = ativa;
    this.inimigo = null;
}

public boolean isAtiva() {
    return this.ativa;
}

public boolean isColisao(Personagem tipo, List<Bolha> bolhas, List<Gota> gotas) {
    if(tipo instanceof PersonagemBlue) {
        this.personagem = (PersonagemBlue) tipo;
    }
}

```

```

else if(tipo instanceof PersonagemRed){
    this.personagem = (PersonagemRed) tipo;
}

bolhas = this.personagem.getBolhas();
gotas = this.personagem.getGotas();

for (int i = 0; i < sizeInimigo(); i++) {
    if (this.personagem.getBounds().intersects(getInimigo(i).getBounds()))
    {
        this.personagem.setVisivel(false);
        deletelnimigo(i);
        this.personagem.decVida(1);

        if(this.personagem.getVida() == 0)
            return true;
    }
}

for (int j = 0; j < bolhas.size() - 1; j++)
    for (int k = 0; k < sizeInimigo(); k++) {
        if
(bolhas.get(j).getBounds().intersects(getInimigo(k).getBounds())) {
            deletelnimigo(k);;
            bolhas.remove(j);
            Control.incPontos(1);
        }

        else if(bolhas.get(j).getX() > 960)
            bolhas.remove(j);
    }

for (int k = 0; k < gotas.size() - 1; k++)
    for (int l = 0; l < sizeInimigo(); l++) {
        if
(gotas.get(k).getBounds().intersects(getInimigo(l).getBounds())) {
            deletelnimigo(l);
            gotas.remove(k);
            Control.incPontos(1);
        }
    }

```

```
        else if(gotas.get(k).getY() > 672)
            gotas.remove(k);
    }

    return false;
}
}
```

ANEXO R – Código da classeFase03.java

```

package br.com.unip.cc.fases;

import java.util.ArrayList;
import java.util.List;

import javax.swing.ImageIcon;

import br.com.unip.cc.armas.Bolha;
import br.com.unip.cc.armas.Gota;
import br.com.unip.cc.base.Personagem;
import br.com.unip.cc.control.Control;
import br.com.unip.cc.inimigos.Inimigo3;
import br.com.unip.cc.personagem.PersonagemBlue;
import br.com.unip.cc.personagem.PersonagemRed;

/**
 * Esta classe manipula os eventos da fase03
 *
 * @version 0.1 29/Ago/2016
 */

public class Fase03 {
    private final ImageIcon REFERENCIA = new ImageIcon("res\\fundojogo3.png");
//pegando a imagem de fundo
    private List<Inimigo3> inimigo = null; //uma lista de objetos do tipo Inimigo3
    private Personagem personagem = null;
    private boolean ativa = false;
    private int[][] coordenadas = {
        {13131, 050}, {10101, 150}, {5555, 200}, {2588, 250},
        {3555, 350}, {4878, 450}, {12590, 530}, {11250, 130},
        {3554, 060}, {3789, 290}, {2547, 050}, {8897, 150},
        {9877, 330}, {21085, 430}, {12254, 500}, {9991, 400},
        {7547, 200}, {5311, 250}, {12455, 450}, {2710, 450},
        {4778, 230}, {2155, 290}, {9157, 070}, {6478, 040},
        {8711, 500}, {21214, 400}, {12001, 530}, {8555, 90},
    };

```

```

public Imagem getReferencia() {
    return REFERENCIA;
}

public Inimigo3 getInimigo(int i) {
    return this.inimigo.get(i);
}

public int sizeInimigo() {
    return this.inimigo.size();
}

public void deleteInimigo(int i) {
    this.inimigo.remove(i);
}

public void inicializaInimigo3() {
    this.inimigo = new ArrayList<Inimigo3>();//objeto que aponta para uma lista de
inimigos

    for (int i = 0; i < this.coordenadas.length; i++)//adicionando inimigos a lista
        this.inimigo.add(new Inimigo3(this.coordenadas[i][0],
this.coordenadas[i][1]));
    this.ativa = true;
}

public void setAtiva(boolean ativa) {
    this.ativa = ativa;
    this.inimigo = null;
}

public boolean isAtiva() {
    return this.ativa;
}

public boolean isColisao(Personagem tipo, List<Bolha> bolhas, List<Gota> gotas) {
    if(tipo instanceof PersonagemBlue) {
        this.personagem = (PersonagemBlue) tipo;
    }
}

```

```

else if(tipo instanceof PersonagemRed){
    this.personagem = (PersonagemRed) tipo;
}

bolhas = this.personagem.getBolhas();
gotas = this.personagem.getGotas();

for (int i = 0; i < sizeInimigo(); i++) {
    if (this.personagem.getBounds().intersects(getInimigo(i).getBounds()))
    {
        this.personagem.setVisivel(false);
        deleteInimigo(i);
        this.personagem.decVida(1);

        if(this.personagem.getVida() == 0)
            return true;
    }
}

for (int j = 0; j < bolhas.size() - 1; j++)
    for (int k = 0; k < sizeInimigo(); k++) {
        if
(bolhas.get(j).getBounds().intersects(getInimigo(k).getBounds())) {
            deleteInimigo(k);
            bolhas.remove(j);
            Control.incPontos(1);
        }

        else if(bolhas.get(j).getX() > 960)
            bolhas.remove(j);
    }

for (int k = 0; k < gotas.size() - 1; k++)
    for (int l = 0; l < sizeInimigo(); l++) {
        if
(gotas.get(k).getBounds().intersects(getInimigo(l).getBounds())) {
            deleteInimigo(l);
            gotas.remove(k);
            Control.incPontos(1);
        }
    }

```

```
        else if(gotas.get(k).getY() > 672)
            gotas.remove(k);
    }

    return false;
}
}
```


ANEXO S – Código da classeFase04.java

```

package br.com.unip.cc.fases;

import java.util.ArrayList;
import java.util.List;

import javax.swing.ImageIcon;

import br.com.unip.cc.armas.Bolha;
import br.com.unip.cc.armas.Gota;
import br.com.unip.cc.base.Personagem;
import br.com.unip.cc.control.Control;
import br.com.unip.cc.inimigos.Inimigo4;
import br.com.unip.cc.personagem.PersonagemBlue;
import br.com.unip.cc.personagem.PersonagemRed;

/**
 * Esta classe manipula os eventos da fase04
 *
 * @version 0.1 29/Ago/2016
 */

public class Fase04 {
    private final ImageIcon REFERENCIA = new ImageIcon("res\\fundojogo4.png");
//pegando a imagem de fundo
    private List<Inimigo4> inimigo = null; //uma lista de objetos do tipo Inimigo4
    private Personagem personagem = null;
    private boolean ativa = false;
    private int[][] coordenadas = {
        {2510, 050}, {3221, 150}, {5110, 200}, {4759, 250},
        {3000, 350}, {6785, 450}, {9958, 530}, {3584, 130},
        {2651, 330}, {5547, 430}, {9994, 500}, {6871, 400},
        {2121, 050}, {3581, 150}, {10258, 060}, {8447, 290},
        {7710, 200}, {6791, 250}, {13013, 450}, {12240, 450},
        {6411, 230}, {2997, 290}, {20910, 070}, {10840, 040},
        {4451, 500}, {2820, 400}, {7611, 530}, {6714, 90},
    };

```

```

public Imagem getReferencia() {
    return REFERENCIA;
}

public Inimigo4 getInimigo(int i) {
    return this.inimigo.get(i);
}

public int sizeInimigo() {
    return this.inimigo.size();
}

public void deleteInimigo(int i) {
    this.inimigo.remove(i);
}

public void inicializaInimigo4() {
    this.inimigo = new ArrayList<Inimigo4>(); //objeto que aponta para uma lista
de inimigos

    for (int i = 0; i < this.coordenadas.length; i++) //adicionando inimigos a lista
        this.inimigo.add(new Inimigo4(this.coordenadas[i][0],
this.coordenadas[i][1]));
    this.ativa = true;
}

public void setAtiva(boolean ativa) {
    this.ativa = ativa;
    this.inimigo = null;
}

public boolean isAtiva() {
    return this.ativa;
}

public boolean isColisao(Personagem tipo, List<Bolha> bolhas, List<Gota> gotas) {
    if(tipo instanceof PersonagemBlue) {
        this.personagem = (PersonagemBlue) tipo;
    }
}

```

```

else if(tipo instanceof PersonagemRed){
    this.personagem = (PersonagemRed) tipo;
}

bolhas = this.personagem.getBolhas();
gotas = this.personagem.getGotas();

for (int i = 0; i < sizeInimigo(); i++) {
    if (this.personagem.getBounds().intersects(getInimigo(i).getBounds()))
    {
        this.personagem.setVisivel(false);
        deleteInimigo(i);
        this.personagem.decVida(1);

        if(this.personagem.getVida() == 0)
            return true;
    }
}

for (int j = 0; j < bolhas.size() - 1; j++)
    for (int k = 0; k < sizeInimigo(); k++) {
        if
(bolhas.get(j).getBounds().intersects(getInimigo(k).getBounds())) {
            deleteInimigo(k);;
            bolhas.remove(j);
            Control.incPontos(1);
        }

        else if(bolhas.get(j).getX() > 960)
            bolhas.remove(j);
    }

for (int k = 0; k < gotas.size() - 1; k++)
    for (int l = 0; l < sizeInimigo(); l++) {
        if
(gotas.get(k).getBounds().intersects(getInimigo(l).getBounds())) {
            deleteInimigo(l);
            gotas.remove(k);
            Control.incPontos(1);
        }
    }

```

```
        else if(gotas.get(k).getY() > 672)
            gotas.remove(k);
    }

    return false;
}
}
```

ANEXO T – Código da classe Fase05.java

```

package br.com.unip.cc.fases;

import java.util.ArrayList;
import java.util.List;

import javax.swing.ImageIcon;

import br.com.unip.cc.armas.Bolha;
import br.com.unip.cc.armas.Gota;
import br.com.unip.cc.base.Personagem;
import br.com.unip.cc.control.Control;
import br.com.unip.cc.inimigos.Boss;
import br.com.unip.cc.personagem.PersonagemBlue;
import br.com.unip.cc.personagem.PersonagemRed;

/**
 * Esta classe manipula os eventos da fase05
 *
 * @version 0.1 29/Ago/2016
 */

public class Fase05 {
    private final ImageIcon REFERENCIA = new ImageIcon("res\\fundojogo5.png");
//pegando a imagem de fundo
    private List<Boss> boss = null; //uma lista de objetos do tipo Boss
    private Personagem personagem = null;
    private boolean ativa = false;
    private int[][] coordenadas = {{2520, 190}};

    public ImageIcon getReferencia() {
        return REFERENCIA;
    }

    public Boss getInimigo(int i) {
        return this.boss.get(i);
    }
}

```

```

public int sizeInimigo() {
    return this.boss.size();
}

public void deleteInimigo(int i) {
    this.boss.remove(i);
}

public void inicializaBoss() {
    this.boss = new ArrayList<Boss>(); //objeto que aponta para uma lista de
inimigos

    for (int i = 0; i < 1; i++) //adicionando inimigos a lista
        this.boss.add(new Boss(this.coordenadas[i][0], this.coordenadas[i][1],
100));

    this.ativa = true;
}

public void setAtiva(boolean ativa) {
    this.ativa = ativa;
    this.boss = null;
}

public boolean isAtiva() {
    return this.ativa;
}

public boolean isColisao(Personagem tipo, List<Bolha> bolhas, List<Gota> gotas) {
    if(tipo instanceof PersonagemBlue) {
        this.personagem = (PersonagemBlue) tipo;
    }

    else if(tipo instanceof PersonagemRed){
        this.personagem = (PersonagemRed) tipo;
    }

    bolhas = this.personagem.getBolhas();
    gotas = this.personagem.getGotas();

    for (int i = 0; i < sizeInimigo(); i++) {

```

```

        if (this.personagem.getBounds().intersects(getInimigo(i).getBounds()))
    {
        this.personagem.setVisivel(false);
        getInimigo(i).setVisivel(false);
        return true;
    }
}

for (int j = 0; j < bolhas.size() - 1; j++)
    for (int k = 0; k < sizeInimigo(); k++) {
        if
(bolhas.get(j).getBounds().intersects(getInimigo(k).getBounds())) {
            Boss.dncVida(1);
            getInimigo(k).setVisivel(false);
            bolhas.remove(j);
            Control.incPontos(1);
        }

        else if(bolhas.get(j).getX() > 960)
            bolhas.remove(j);
    }

for (int k = 0; k < gotas.size() - 1; k++)
    for (int l = 0; l < sizeInimigo(); l++) {
        if
(gotas.get(k).getBounds().intersects(getInimigo(l).getBounds())) {
            gotas.remove(k);
        }

        else if(gotas.get(k).getY() > 672)
            gotas.remove(k);
    }
return false;
}
}

```

ANEXO U – Código da classe Boss.java

```

package br.com.unip.cc.inimigos;

import javax.swing.ImageIcon;

import br.com.unip.cc.base.ImageEntity;

/**
 * Esta classe representa o chefe e é uma subclasse de ImageEntity
 *
 * @version 0.1 29/Ago/2016
 */

public class Boss extends ImageEntity {
    private static final int VELOCIDADE = 1;
    private static int vida = 0;

    /**
     * ao instanciar o chefe, o construtor busca todas as imagens localizadas na
     pasta res
     * recebe três parâmetros, dos quais dois instanciarão o objeto em X e Y e um é
     a vida inicial
     *
     * @param x
     * @param y
     * @param vida
     */
    public Boss(int x, int y, int vida) {
        ImageIcon referencia;

        referencia = new ImageIcon("res\\Boss1.png");
        image = referencia.getImage();

        setX(x);
        setY(y);
        setVida(vida);
        setVisivel(true);
    }

```



```
public void mexer() {  
    if(getX() < -500)  
        setX(LARGURA_TELA + 1003);  
  
    else  
        incX(-VELOCIDADE);  
}  
  
public static int getVida() {  
    return vida;  
}  
  
public static void setVida(int vida) {  
    Boss.vida = vida;  
}  
  
public static void dncVida(int vida) {  
    Boss.vida -= vida;  
}  
}
```

ANEXO V – Código da classe Inimigo1.java

```

package br.com.unip.cc.inimigos;

import javax.swing.ImageIcon;

import br.com.unip.cc.base.ImageEntity;

/**
 * Esta classe representa o inimigo 1 e é uma subclade de ImageEntity
 *
 * @version 0.1 29/Ago/2016
 */

public class Inimigo1 extends ImageEntity {
    private static final int VELOCIDADE = 2;
    private static int contador = 0;

    /**
     * ao instanciar o inimigo 1, o cosntrutor busca todas as imagens localizadas na pasta
     *
     * recebe dois parametros, do qual instanceara o objeto em X e Y
     *
     * @param x
     * @param y
     */
    public Inimigo1(int x, int y) {
        ImageIcon referencia;

        setX(x);
        setY(y);

        if(contador++ % 3 == 0)
            referencia = new ImageIcon("res\\inimigo1.png");

        else
            referencia = new ImageIcon("res\\inimigo11.png");

        image = referencia.getImage();
    }

```

```
        setVisivel(true);
    }

    public void mexer() {
        if(getX() < 0)
            setX(LARGURA_TELA + 3);

        else
            incX(-VELOCIDADE);
    }
}
```

ANEXO W – Código da classe Inimigo2.java

```

package br.com.unip.cc.inimigos;

import javax.swing.ImageIcon;

import br.com.unip.cc.base.ImageEntity;

/**
 * Esta classe representa o inimigo 2 e é uma subclade de ImageEntity
 *
 * @version 0.1 29/Ago/2016
 */

public class Inimigo2 extends ImageEntity {
    private static final int VELOCIDADE = 3;
    private static int contador = 0;

    /**
     * ao instanciar o inimigo 2, o cosntrutor busca todas as imagens localizadas na pasta
     *
     * recebe dois parametros, do qual instanceara o objeto em X e Y
     *
     * @param x
     * @param y
     */
    public Inimigo2(int x, int y) {
        ImageIcon referencia;

        setX(x);
        setY(y);

        if(contador++ % 3 == 0)
            referencia = new ImageIcon("res\\inimigo2.png");

        else
            referencia = new ImageIcon("res\\inimigo22.png");

        image = referencia.getImage();
    }
}

```

```
        setVisivel(true);
    }

    public void mexer() {
        if(getX() < 0)
            setX(LARGURA_TELA + 3);

        else
            incX(-VELOCIDADE);
    }
}
```

ANEXO X – Código da classe Inimigo3.java

```

package br.com.unip.cc.inimigos;

import javax.swing.ImageIcon;

import br.com.unip.cc.base.ImageEntity;

/**
 * Esta classe representa o inimigo 2 e é uma subclade de ImageEntity
 *
 * @version 0.1 29/Ago/2016
 */

public class Inimigo3 extends ImageEntity {
    private static final int VELOCIDADE = 3;
    private static int contador = 0;

    /**
     * ao instanciar o inimigo 3, o cosntrutor busca todas as imagens localizadas na pasta
     *
     * recebe dois parametros, do qual instanceara o objeto em X e Y
     *
     * @param x
     * @param y
     */
    public Inimigo3(int x, int y) {
        ImageIcon referencia;

        setX(x);
        setY(y);

        if(contador++ % 3 == 0)
            referencia = new ImageIcon("res\\inimigo3.png");

        else
            referencia = new ImageIcon("res\\inimigo33.png");

        image = referencia.getImage();
    }
}

```

```
        setVisivel(true);
    }

    public void mexer() {
        if(getX() < 0)
            setX(LARGURA_TELA + 3);

        else
            incX(-VELOCIDADE);
    }
}
```

ANEXO Y – Código da classe Inimigo4.java

```

package br.com.unip.cc.inimigos;

import javax.swing.ImageIcon;

import br.com.unip.cc.base.ImageEntity;

/**
 * Esta classe representa o inimigo 2 e é uma subclade de ImageEntity
 *
 * @version 0.1 29/Ago/2016
 */

public class Inimigo4 extends ImageEntity {
    private static final int VELOCIDADE = 4;

    /**
     * ao instanciar o inimigo 4, o cosntrutor busca todas as imagens localizadas na pasta
     *
     * recebe dois parametros, do qual instanceara o objeto em X e Y
     *
     * @param x
     * @param y
     */
    public Inimigo4(int x, int y) {
        ImageIcon referencia;

        setX(x);
        setY(y);

        referencia = new ImageIcon("res\\inimigo4.png");

        image = referencia.getImage();
        setVisivel(true);
    }

    public void mexer() {
        if(getX() < 0)

```



```
        setX(LARGURA_TELA + 3);  
  
    else  
        incX(-VELOCIDADE);  
    }  
}
```

ANEXO Z – Código da classe JFase.java

```
package br.com.unip.cc.janelas;

import javax.swing.JFrame;

import br.com.unip.cc.control.Fase;

/**
 * Esta classe a troca de fase do jogo
 *
 * @version 0.1 09/Set/2016
 */

public class JFase extends JFrame{
    private static final long serialVersionUID = 1L;
    private Fase painel = null;

    public JFase() {
        super();
        initialize();
        setVisible(true);
    }

    public void initialize() {
        painel = new Fase();
        painel.setBounds(0, 0, 997, 672);
        this.setSize(997,672);
        this.setTitle("Alpha Gênesis");
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setBounds(150, 150, 997,672);
        this.setVisible(true);
        this.setResizable(false);
        this.setLocationRelativeTo(null);
        this.getContentPane().setLayout(null);
        this.getContentPane().add(painel);
    }
}
```

ANEXO AA – Código da classe JFBatalha.java

```
package br.com.unip.cc.janelas;

import javax.swing.JFrame;

import br.com.unip.cc.control.Batalha;

/**
 * Esta classe representa o campo de batalha do jogo
 *
 * @version 0.1 09/Set/2016
 */

public class JFBatalha extends JFrame{
    private static final long serialVersionUID = 1L;
    private Batalha painel = null;

    public JFBatalha() {
        super();
        initialize();
        setVisible(true);
    }

    public void initialize() {
        painel = new Batalha();
        painel.setBounds(0, 0, 997, 672);
        this.setSize(997,672);
        this.setTitle("Batalha Gênesis");
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setBounds(150, 150, 997,672);
        this.setVisible(true);
        this.setResizable(false);
        this.setLocationRelativeTo(null);
        this.getContentPane().setLayout(null);
        this.getContentPane().add(painel);
    }
}
```

ANEXO AB – Código da classe JFHome.java

```
package br.com.unip.cc.janelas;

import java.awt.Rectangle;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JPanel;

import br.com.unip.cc.music.TocarSom;

/**
 * Esta classe representa o menu do jogo
 *
 * @version 0.1 02/Set/2016
 */

public class JFHome extends JFrame{
    private static final long serialVersionUID = 1L;
    private JPanel jContentPane = null;
    private JLabel lblMenu = null;
    private JButton btnJogar = null;
    private JButton btnRecordes = null;
    private JButton btnSobre = null;
    private JButton btnSair = null;

    public JFHome() {
        super();
        initialize();
        TocarSom.som("Game_music");
    }

    public void initialize() {
```

```

        this.setSize(997,672);
        this.setContentPane(getJContentPane());
        this.setTitle("Menu");
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setBounds(150, 150, 997,672);
        this.setVisible(true);
        this.setResizable(false);
        this.setLocationRelativeTo(null);
        this.getContentPane().setLayout(null);
    }

    private JPanel getJContentPane() {
        if (jContentPane == null) {
            lblMenu = new JLabel("");
            lblMenu.setBounds(0, 0, 997, 672); //posição da imagem
            lblMenu.setIcon(new ImageIcon("res//tela_principal2.png")); //imagem

            jContentPane = new JPanel();
            jContentPane.setLayout(null);
            jContentPane.add(lblMenu, null);
            jContentPane.add(getBtnJogar(), null);
            jContentPane.add(getBtnRecordes(), null);
            jContentPane.add(getBtnSobre(), null);
            jContentPane.add(getBtnSair(), null);
        }

        return jContentPane;
    }

    private JButton getBtnJogar() {
        if(btnJogar == null) {
            btnJogar = new JButton();
            btnJogar.setBounds(new Rectangle(320, 180, 362, 72));
            btnJogar.setText("");
            btnJogar.setIcon(new ImageIcon("res//iniciar.png"));
            btnJogar.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent arg0) {
                    setVisible(false);
                    @SuppressWarnings("unused")
                    JFModo modo = new JFModo();
                }
            });
        }
    }

```

de fundo

```

        }
    });
}

return btnJogar;
}

private JButton getBtnRecordes() {
    if(btnRecordes == null) {
        btnRecordes = new JButton();
        btnRecordes.setBounds(320, 300, 362, 72);
        btnRecordes.setIcon(new ImageIcon("res//Recordes.png"));
        btnRecordes.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent arg0) {
                @SuppressWarnings("unused")
                JFRecordes recordes = new JFRecordes();
            }
        });
    }
    return btnRecordes;
}

private JButton getBtnSobre() {
    if(btnSobre == null) {
        btnSobre = new JButton();
        btnSobre.setBounds(new Rectangle(320, 420, 362, 72));
        btnSobre.setText("");
        btnSobre.setIcon(new ImageIcon("res//sobre.png"));
        btnSobre.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) {
                JOptionPane.showMessageDialog(null, "- Gabriel de
Almeida Batista\n- Felipe da Silva Borges Neves\n- Felipe Cunha Santos\n- José Vitor Zanonni da
Costa", "Equipe de Desenvolvimento", JOptionPane.PLAIN_MESSAGE);
            }
        });
    }
    return btnSobre;
}

```

```
private JButton getBtnSair() {  
    if(btnSair == null) {  
        btnSair = new JButton();  
        btnSair.setBounds(new Rectangle(320, 540, 362, 72));  
        btnSair.setText("");  
        btnSair.setIcon(new ImageIcon("res//sair.png"));  
        btnSair.addActionListener(new ActionListener() {  
            public void actionPerformed(ActionEvent e) {  
                System.exit(0);  
            }  
        });  
    }  
    return btnSair;  
}  
}
```

ANEXO AC – Código da classe JFModo.java

```
package br.com.unip.cc.janelas;

import java.awt.Rectangle;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;

import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JFrame;
import javax.swing.JLabel;
import javax.swing.JPanel;

/**
 * Esta classe representa a escolha de modo de jogo
 *
 * @version 0.1 11/Set/2016
 */

public class JFModo extends JFrame{
    private static final long serialVersionUID = 1L;
    private JPanel jContentPane = null;
    private JLabel lblModo = null;
    private JButton btnHistoria = null;
    private JButton btnBatalha = null;

    public JFModo() {
        super();
        initialize();
    }

    public void initialize() {
        this.setSize(997,672);
        this.setContentPane(getJContentPane());
        this.setTitle("Modo de Jogo");
        this.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        this.setBounds(150, 150, 997,672);
        this.setVisible(true);
    }
}
```



```

        this.setResizable(false);
        this.setLocationRelativeTo(null);
        this.getContentPane().setLayout(null);
    }

    private JPanel getJContentPane() {
        if (jContentPane == null) {
            lblModo = new JLabel("");
            lblModo.setBounds(0, 0, 997, 672); //posição da imagem
            lblModo.setIcon(new ImageIcon("res//tela_principal2.png")); //imagem

            jContentPane = new JPanel();
            jContentPane.setLayout(null);
            jContentPane.add(lblModo, null);
            jContentPane.add(getBtnHistoria(), null);
            jContentPane.add(getBtnRecordes(), null);
        }

        return jContentPane;
    }

    private JButton getBtnHistoria() {
        if(btnHistoria == null) {
            btnHistoria = new JButton();
            btnHistoria.setBounds(new Rectangle(320, 180, 362, 72));
            btnHistoria.setText("");
            btnHistoria.setIcon(new ImageIcon("res//Hist.png"));
            btnHistoria.addActionListener(new ActionListener() {
                public void actionPerformed(ActionEvent arg0) {
                    setVisible(false);
                    @SuppressWarnings("unused")
                    JFase fases = new JFase();
                }
            });
        }

        return btnHistoria;
    }

    private JButton getBtnRecordes() {

```

de fundo

```
if(btnBatalha == null) {  
    btnBatalha = new JButton();  
    btnBatalha.setBounds(320, 300, 362, 72);  
    btnBatalha.setIcon(new ImageIcon("res//P1VP2.png"));  
    btnBatalha.addActionListener(new ActionListener() {  
        public void actionPerformed(ActionEvent arg0) {  
            setVisible(false);  
            @SuppressWarnings("unused")  
            JFBatalha batalha = new JFBatalha();  
        }  
    });  
}  
return btnBatalha;  
}  
}
```

ANEXO AD – Código da classe JFRecordes.java

```
package br.com.unip.cc.janelas;

import java.sql.ResultSet;
import java.sql.SQLException;

import javax.swing.DefaultListModel;
import javax.swing.JFrame;
import javax.swing.JList;
import javax.swing.JPanel;
import javax.swing.JScrollPane;
import javax.swing.border.EmptyBorder;

import br.com.unip.cc.jdbc.JogadorDAO;
import javax.swing.JLabel;
import java.awt.Font;
import java.awt.Color;

/**
 * Esta classe representa o painel de recordes
 *
 * @version 0.1 09/Set/2016
 */

public class JFRecordes extends JFrame {
    private static final long serialVersionUID = 1L;
    private JPanel contentPane;
    private JScrollPane scrollPane;
    @SuppressWarnings("rawtypes")
    private JList listRecordes;

    @SuppressWarnings({ "rawtypes", "unchecked" })
    public JFRecordes() {
        setTitle("Recordes");
        setDefaultCloseOperation(JFrame.HIDE_ON_CLOSE);
        setBounds(100, 100, 354, 223);
        setVisible(true);
        setResizable(false);
    }
}
```

```

        contentPane = new JPanel();
        contentPane.setBorder(new EmptyBorder(5, 5, 5, 5));
        setContentPane(contentPane);
        contentPane.setLayout(null);

        listRecordes = new JList(listaJogador());

        scrollPane = new JScrollPane(listRecordes);
        scrollPane.setBounds(5, 35, 340, 150);
        contentPane.add(scrollPane);

        JLabel lblRecordes = new JLabel("Lista de Recordes");
        lblRecordes.setForeground(Color.RED);
        lblRecordes.setFont(new Font("Arial", Font.BOLD, 13));
        lblRecordes.setBounds(125, 11, 119, 14);
        contentPane.add(lblRecordes);
    }

    @SuppressWarnings({ "rawtypes", "unchecked" })
    public DefaultListModel listaJogador() {
        try {
            DefaultListModel lista = new DefaultListModel();
            JogadorDAO player = new JogadorDAO();
            ResultSet rs = player.pesquisar();

            for(int i = 0; i < 10; i++)
                if(rs.next())
                    lista.addElement(rs.getString(1) + " " + rs.getString(2) + " " + rs.getString(3));
            return lista;
        } catch (SQLException e) {
            e.printStackTrace();
            return null;
        }
    }
}

```

ANEXO AE – Código da classe Conexao.java

```

package br.com.unip.cc.jdbc;

import java.sql.Connection;
import java.sql.DriverManager;
import java.sql.SQLException;

import javax.swing.JOptionPane;

/**
 * Esta classe cuida da conexao com o SGBD Oracle 11g
 *
 * @version 0.1 09/Set/2016
 */

public class Conexao {
    private static String url = "jdbc:oracle:thin:@127.0.0.1:1521:XE"; //url do banco local
    private static String login = "GABRIEL"; //usuario do banco
    private static String senha = "gabriel2b21"; //senha do usuario
    private static Connection con = null; //objeto que aponta para a conexao

    public static Connection getConexao() {
        if(con == null){// caso nao exista uma conexao
            try{
                Class.forName("oracle.jdbc.driver.OracleDriver");

                //recuperando o driver do oracle
                con = DriverManager.getConnection(url, login, senha);

                //conectando ao banco

            } catch(ClassNotFoundException e) { //caso nao encontre o driver
                JOptionPane.showMessageDialog(null, "Classe do Driver de
conexao com Oracle não encontrada!", e.getMessage(), JOptionPane.ERROR_MESSAGE);
            } catch(SQLException e) { //caso a url ou o login ou a senha esteja
incorreto
                JOptionPane.showMessageDialog(null, "Problemas com
Parâmetros da conexão!", e.getMessage(), JOptionPane.ERROR_MESSAGE);
            }
        }
    }
}

```

```
        return con;
    }

    public static void closeConexao() {
        try{
            con.close();
        } catch (SQLException e){
            JOptionPane.showMessageDialog(null, "Erro:Não foi possível fechar
a conexão com BD!");
        }
    }
}
```

ANEXO AF – Código da classe Jogador.java

```
package br.com.unip.cc.jdbc;

import java.util.Calendar;

/**
 * Esta classe cuida de capturar as informacoes do jogador
 *
 * @version 0.1 09/Set/2016
 */

public class Jogador {
    private String nome = null;
    private String nick = null;
    private int record = 0;
    private Calendar data = Calendar.getInstance();

    public String getNome() {
        return nome;
    }

    public void setNome(String nome) {
        this.nome = nome;
    }

    public String getNick() {
        return nick;
    }

    public void setNick(String nick) {
        this.nick = nick;
    }

    public int getRecord() {
        return record;
    }

    public void setRecord(int codigo) {
```

```
        this.record = codigo;
    }

    public Calendar getData() {
        return data;
    }

    public void setData(Calendar data) {
        this.data = data;
    }
}
```


ANEXO AG – Código da classe JogadorDAO.java

```

package br.com.unip.cc.jdbc;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.ResultSet;
import java.sql.SQLException;
import java.util.Calendar;

/**
 * Esta classe cuida de inserir e recuperar jogadores no banco
 *
 * @version 0.1 09/Set/2016
 */

public class JogadorDAO {
    private Connection con = null;
    private PreparedStatement stmt = null;
    private String sql = null;
    /**
     * metodo que insere um novo jogador na tabela
     * @param player
     * @return
     */
    public String inserirPlayer(Jogador player) {
        con = Conexao.getConexao();
        sql = "INSERT INTO JOGADORES VALUES(?, ?, ?, ?)";
        try {
            stmt = con.prepareStatement(sql);
            stmt.setString(1, "Inc_JOG.NEXTVAL");
            stmt.setString(2, player.getNome());
            stmt.setString(3, player.getNick());
            stmt.setString(4, player.getData().get(Calendar.DAY_OF_MONTH) +
"/" + player.getData().get(Calendar.MONTH) + "/" + player.getData().get(Calendar.YEAR));
            stmt.execute();

            return "Inserido com sucesso!";
        } catch(SQLException e) {

```

```

        return e.getMessage();
    }
}

/**
 * metodo que recupera os maiores recordes
 * @return
 */
public ResultSet pesquisar() {
    con = Conexao.getConexao();
    sql = "SELECT J.JOG_NOME, J.JOG_APELIDO, R.REC_QUANTIDADE
FROM JOGADORES J, RECORDES R WHERE J.JOG_CODIGO = R.REC_CODIGO ORDER BY
R.REC_QUANTIDADE DESC";

    try {
        PreparedStatement stmt = con.prepareStatement(sql);
        ResultSet rs = stmt.executeQuery();
        return rs;
    } catch(SQLException e) {
        e.printStackTrace();
        return null;
    }
}
}

```

ANEXO AH – Código da classe RecordeDAO.java

```

package br.com.unip.cc.jdbc;

import java.sql.Connection;
import java.sql.PreparedStatement;
import java.sql.SQLException;
import java.util.Calendar;

/**
 * Esta classe cuida de inserir e recuperar jogadores no banco
 *
 * @version 0.1 07/Nov/2016
 */

public class RecordeDAO {
    private Connection con = null;
    private PreparedStatement stmt = null;
    private String sql = null;
    /**
     * metodo que insere um novo jogador na tabela
     * @param player
     * @return
     */

    public String inserirRecorde(Jogador player){
        con = Conexao.getConexao();
        sql = "INSERT INTO RECORDES VALUES(?, ?, ?, ?)";
        try {
            stmt = con.prepareStatement(sql);
            stmt.setString(1, "INC_REC.NEXTVAL");
            stmt.setString(2, "INC_JOG1.NEXTVAL");
            stmt.setString(3, player.getData().get(Calendar.DAY_OF_MONTH) +
"/" + player.getData().get(Calendar.MONTH) + "/" + player.getData().get(Calendar.YEAR));
            stmt.setInt(4, player.getRecord());
            stmt.execute();

            return "Inserido com sucesso!";
        } catch(SQLException e) {

```

```
        return e.getMessage();  
    }  
}  
}
```

ANEXO AI – Código da classe TocarSom.java

```

package br.com.unip.cc.music;

import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.io.FileInputStream;
import java.io.IOException;

import sun.audio.AudioPlayer;
import sun.audio.AudioStream;

/**
 * Esta classe controla os audios do jogo
 *
 * @version 0.1 11/Set/2016
 */

public class TocarSom implements ActionListener {
    public final void actionPerformed(ActionEvent e) {
        som("");
    }

    public static void som(String musica) {
        AudioPlayer MGP = AudioPlayer.player;
        AudioStream BGM = null;
        try {
            BGM = new AudioStream(
                new FileInputStream("res/" + musica + ".wav"));
        } catch (IOException e) {
            System.out.println("ERRO!");
        }
        MGP.start(BGM);
    }
}

```

ANEXO AJ – Código da classe EntityManagerProvider.java

```
package br.com.unip.cc.persistence;

import javax.persistence.EntityManager;
import javax.persistence.EntityManagerFactory;
import javax.persistence.Persistence;

/**
 * Classe responsável pela manipulação dos objetos que devem ser salvos e recuperados do
 banco de dados
 *
 * @version 0.1 16/Out/2016
 */

public class EntityManagerProvider {
    private static EntityManagerFactory emf = null;

    private EntityManagerProvider() {
    }

    public static EntityManager getEntityManagerInstance() {
        if (emf == null)
            emf = Persistence.createEntityManagerFactory("conexao");

        return emf.createEntityManager();
    }
}
```

ANEXO AK – Código da classe PersonagemBlue.java

```

package br.com.unip.cc.personagem;

import java.awt.event.KeyEvent;
import java.util.ArrayList;
import java.util.List;

import javax.swing.ImageIcon;

import br.com.unip.cc.armas.Bolha;
import br.com.unip.cc.armas.Gota;
import br.com.unip.cc.base.ImageEntity;
import br.com.unip.cc.base.Personagem;
import br.com.unip.cc.music.TocarSom;

/**
 * Esta classe representa o personagem principal
 *
 * @version 0.1 02/Set/2016
 */

public class PersonagemBlue extends ImageEntity implements Personagem{
    private ImageIcon cima = null;
    private ImageIcon ref = null;
    private ImageIcon baixo = null;
    private ImageIcon esquerda = null;
    private ImageIcon direita = null;
    private ImageIcon anim_L = null;
    private ImageIcon anim_R = null;
    private ImageIcon anim_U = null;
    private ImageIcon anim_D = null;
    private List<Bolha> bolhas = null;
    private List<Gota> gotas = null;
    private boolean batalha = false;
    private int dx = 0, dy = 0; //direcao de x e y
    private int vida = 0;

    /**

```

* ao instanciar o personagem, ele já inicia com os seus atributos preenchidos
*/

```
public PersonagemBlue(boolean batalha) {
    baixo = new ImagemIcon("res\\amena4.png");
    esquerda = new ImagemIcon("res\\amena2.png");
    direita = new ImagemIcon("res\\amena1.png");
    cima = new ImagemIcon("res\\amena3.png");
    anim_L = new ImagemIcon("res\\Animation_L.gif");
    anim_R = new ImagemIcon("res\\Animation_R.gif");
    anim_U = new ImagemIcon("res\\Animation_U.gif");
    anim_D = new ImagemIcon("res\\Animation_D.gif");
    ref = new ImagemIcon("res\\ref.png");
    image = ref.getImage();
    bolhas = new ArrayList<Bolha>();
    gotas = new ArrayList<Gota>();
    setBatalha(batalha);
    setX(50);
    setY(50);

    if(isBatalha())
        setVida(100);

    else
        setVida(3);
}

public void mexer() {
    incX(dx);
    incY(dy);

    if(getX() < 1)
        setX(1);

    else if(getX() > 930)
        setX(930);

    else if(getY() < 1)
        setY(1);

    else if(getY() > 580)
```



```

        setY(580);
    }

    public List<Bolha> getBolhas() {
        return bolhas;
    }

    public Bolha getBolhas(int i) {
        return bolhas.get(i);
    }

    public int sizeBolhas() {
        return bolhas.size();
    }

    public List<Gota> getGotas() {
        return gotas;
    }

    public Gota getGotas(int i) {
        return gotas.get(i);
    }

    public int sizeGotas() {
        return gotas.size();
    }

    public void atira() {
        this.bolhas.add(new Bolha((int)getX() + getWidth(), (int)getY() +
getHeight()/2));
    }

    public void atira2() {
        if(this.gotas.size() < 5){
            this.gotas.add(new Gota((int)getX() + getWidth(), (int)getY() +
getHeight()/2));

            TocarSom.som("Gota");
        }
    }
}

```

```

//Quando o jogador pressionar alguma tecla
public void keyPressed(KeyEvent tecla) {
    int codigo = tecla.getKeyCode();

    if(isBatalha())
        switch(codigo) {
            case KeyEvent.VK_W:
                dy = -1;
                image = anim_U.getImage();
                break;

            case KeyEvent.VK_S:
                dy = 1;
                image = anim_D.getImage();
                break;

            case KeyEvent.VK_A:
                dx = -1;
                image = anim_L.getImage();
                break;

            case KeyEvent.VK_D:
                dx = 1;
                image = anim_R.getImage();
                break;
        }

    else {
        if((codigo == KeyEvent.VK_UP) || (codigo == KeyEvent.VK_W)) {
            dy = -1;
            image = anim_U.getImage();
        }

        else if((codigo == KeyEvent.VK_DOWN) || (codigo ==
KeyEvent.VK_S)) {
            dy = 1;
            image = anim_D.getImage();
        }
    }
}

```

```

else if((codigo == KeyEvent.VK_LEFT) || (codigo == KeyEvent.VK_A))
{
    dx = -1;
    image = anim_L.getImage();
}

else if((codigo == KeyEvent.VK_RIGHT) || (codigo ==
KeyEvent.VK_D)){
    dx = 1;
    image = anim_R.getImage();
}
}
}

```

//Quando o jogador soltar alguma tecla

```
public void keyReleased(KeyEvent tecla) {
```

```
    int codigo = tecla.getKeyCode();
```

```
    if(isBatalha())
```

```
        switch(codigo) {
```

```
        case KeyEvent.VK_J:
```

```
            if(bolhas.size() < 20) {
```

```
                atira();
```

```
                TocarSom.som("Bolha");
```

```
            }
```

```
            break;
```

```
        case KeyEvent.VK_K:
```

```
            atira2();
```

```
            break;
```

```
        case KeyEvent.VK_W:
```

```
            dy = 0;
```

```
            image = cima.getImage();
```

```
            break;
```

```
        case KeyEvent.VK_S:
```

```
            dy = 0;
```

```
            image = baixo.getImage();
```

```
            break;
```

```

        case KeyEvent.VK_A:
            dx = 0;
            image = esquerda.getImage();
            break;

        case KeyEvent.VK_D:
            dx = 0;
            image = direita.getImage();
            break;
    }

    else {
        if(codigo == KeyEvent.VK_J){
            if(bolhas.size() < 20) {
                atira();
                TocarSom.som("Bolha");
            }
        }

        else if(codigo == KeyEvent.VK_K){
            atira2();
        }

        else if((codigo == KeyEvent.VK_UP) || (codigo == KeyEvent.VK_W)) {
            dy = 0;
            image = cima.getImage();
        }

        else if((codigo == KeyEvent.VK_DOWN) || (codigo ==
KeyEvent.VK_S)) {

            dy = 0;
            image = baixo.getImage();
        }

        else if((codigo == KeyEvent.VK_LEFT) || (codigo == KeyEvent.VK_A))
        {

            dx = 0;
            image = esquerda.getImage();
        }
    }

```

```
        else if((codigo == KeyEvent.VK_RIGHT) || (codigo ==
KeyEvent.VK_D)) {
            dx = 0;
            image = direita.getImage();
        }
    }

    public int getVida() {
        return vida;
    }

    public void setVida(int vida) {
        this.vida = vida;
    }

    public void incVida(int vida) {
        this.vida += vida;
    }

    public void decVida(int vida) {
        this.vida -= vida;
    }

    public boolean isBatalha() {
        return batalha;
    }

    public void setBatalha(boolean batalha) {
        this.batalha = batalha;
    }
}
```

ANEXO AL – Código da classe PersonagemRed.java

```
package br.com.unip.cc.personagem;

import java.awt.event.KeyEvent;
import java.util.ArrayList;
import java.util.List;

import javax.swing.ImageIcon;

import br.com.unip.cc.armas.Bolha;
import br.com.unip.cc.armas.Gota;
import br.com.unip.cc.base.ImageEntity;
import br.com.unip.cc.base.Personagem;
import br.com.unip.cc.music.TocarSom;

/**
 * Esta classe representa o personagem secundario
 *
 * @version 0.1 02/Set/2016
 */

public class PersonagemRed extends ImageEntity implements Personagem{
    private ImageIcon cima = null;
    private ImageIcon ref = null;
    private ImageIcon baixo = null;
    private ImageIcon esquerda = null;
    private ImageIcon direita = null;
    private ImageIcon anim_L = null;
    private ImageIcon anim_R = null;
    private ImageIcon anim_U = null;
    private ImageIcon anim_D = null;
    private List<Bolha> bolhas = null;
    private List<Gota> gotas = null;
    private boolean batalha = false;
    private int dx = 0, dy = 0; //direcao de x e y
    private int vida = 0;

    /**
```

* ao instanciar o personagem, ele ja inicia com os seus atributos preenchidos
*/

```
public PersonagemRed(boolean batalha) {
    baixo = new ImagemIcon("res\\geleiaB1.png");
    esquerda = new ImagemIcon("res\\geleiaE1.png");
    direita = new ImagemIcon("res\\geleiaD1.png");
    cima = new ImagemIcon("res\\geleiaC1.png");
    anim_L = new ImagemIcon("res\\geleiaE.gif");
    anim_R = new ImagemIcon("res\\geleiaD.gif");
    anim_U = new ImagemIcon("res\\geleiaC.gif");
    anim_D = new ImagemIcon("res\\geleiaB.gif");
    ref = new ImagemIcon("res\\ref_V.png");
    image = ref.getImage();
    bolhas = new ArrayList<Bolha>();
    gotas = new ArrayList<Gota>();
    setBatalha(batalha);

    if(isBatalha()) {
        setVida(100);
        setX(900);
        setY(550);
    }

    else {
        setVida(3);
        setX(50);
        setY(50);
    }
}

public void mexer() {
    incX(dx);
    incY(dy);

    if(getX() < 1)
        setX(1);

    else if(getX() > 930)
        setX(930);
}
```

```

        else if(getY() < 1)
            setY(1);

        else if(getY() > 580)
            setY(580);
    }

    public List<Bolha> getBolhas() {
        return bolhas;
    }

    public List<Gota> getGotas() {
        return gotas;
    }

    public void atira() {
        this.bolhas.add(new Bolha((int)getX() + getWidth(), (int)getY() +
getHeight()/2));
    }

    public void atira2() {
        if(this.gotas.size() < 5) {
            this.gotas.add(new Gota((int)getX() + getWidth(), (int)getY() +
getHeight()/2));

            TocarSom.som("Gota");
        }
    }

    //Quando o jogador pressionar alguma tecla
    public void keyPressed(KeyEvent tecla) {
        int codigo = tecla.getKeyCode();

        if(isBatalha())
            switch(codigo) {
                case KeyEvent.VK_UP:
                    dy = -1;
                    image = anim_U.getImage();;
                    break;

                case KeyEvent.VK_DOWN:

```



```

        dy = 1;
        image = anim_D.getImage();
        break;

    case KeyEvent.VK_LEFT:
        dx = -1;
        image = anim_L.getImage();
        break;

    case KeyEvent.VK_RIGHT:
        dx = 1;
        image = anim_R.getImage();
        break;
    }

    else {
        if((codigo == KeyEvent.VK_UP) || (codigo == KeyEvent.VK_W)) {
            dy = -1;
            image = anim_U.getImage();
        }

        else if((codigo == KeyEvent.VK_DOWN) || (codigo ==
KeyEvent.VK_S)) {
            dy = 1;
            image = anim_D.getImage();
        }

        else if((codigo == KeyEvent.VK_LEFT) || (codigo == KeyEvent.VK_A))
{
            dx = -1;
            image = anim_L.getImage();
        }

        else if((codigo == KeyEvent.VK_RIGHT) || (codigo ==
KeyEvent.VK_D)){
            dx = 1;
            image = anim_R.getImage();
        }
    }
}
}

```

```

//Quando o jogador soltar alguma tecla
public void keyReleased(KeyEvent tecla) {
    int codigo = tecla.getKeyCode();

    if(isBatalha())
        switch(codigo) {
            case KeyEvent.VK_CONTROL:
                if(bolhas.size() < 20) {
                    atira();
                    TocarSom.som("Bolha");
                }
                break;

            case KeyEvent.VK_SHIFT:
                atira2();
                break;

            case KeyEvent.VK_UP:
                dy = 0;
                image = cima.getImage();
                break;

            case KeyEvent.VK_DOWN:
                dy = 0;
                image = baixo.getImage();
                break;

            case KeyEvent.VK_LEFT:
                dx = 0;
                image = esquerda.getImage();
                break;

            case KeyEvent.VK_RIGHT:
                dx = 0;
                image = direita.getImage();
                break;
        }

    else {

```

```

        if(codigo == KeyEvent.VK_J){
            if(bolhas.size() < 20) {
                atira();
                TocarSom.som("Bolha");
            }
        }

        else if(codigo == KeyEvent.VK_K){
            atira2();
        }

        else if((codigo == KeyEvent.VK_UP) || (codigo == KeyEvent.VK_W)) {
            dy = 0;
            image = cima.getImage();
        }

        else if((codigo == KeyEvent.VK_DOWN) || (codigo ==
KeyEvent.VK_S)) {

            dy = 0;
            image = baixo.getImage();
        }

        else if((codigo == KeyEvent.VK_LEFT) || (codigo == KeyEvent.VK_A))
{

            dx = 0;
            image = esquerda.getImage();
        }

        else if((codigo == KeyEvent.VK_RIGHT) || (codigo ==
KeyEvent.VK_D)) {

            dx = 0;
            image = direita.getImage();
        }
    }

    public int getVida() {
        return vida;
    }

```

```
public void setVida(int vida) {  
    this.vida = vida;  
}  
  
public void incVida(int vida) {  
    this.vida += vida;  
}  
  
public void decVida(int vida) {  
    this.vida -= vida;  
}  
  
public boolean isBatalha() {  
    return batalha;  
}  
  
public void setBatalha(boolean batalha) {  
    this.batalha = batalha;  
}  
}
```

ANEXO AM – Código do XMLpersistence.xml

```

<?xml version="1.0" encoding="UTF-8"?>
<persistence version="2.0"
  xmlns="http://java.sun.com/xml/ns/persistence"
  xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
  xsi:schemaLocation="http://java.sun.com/xml/ns/persistence
    http://java.sun.com/xml/ns/persistence/persistence_2_0.xsd" >

  <persistence-unit name="conexao"><!-- Configura o nome do contexto que
será utilizado -->

    <properties>
      <property name="javax.persistence.jdbc.driver"
value="oracle.jdbc.OracleDriver" /><!-- nome completo da classe do driver JDBC -->
      <property name="javax.persistence.jdbc.user"
value="GABRIEL" /><!-- Usuário do banco de dados -->
      <property name="javax.persistence.jdbc.password"
value="gabriel2b21" /><!-- Senha do banco de dados -->
      <property name="javax.persistence.jdbc.url"
value="jdbc:oracle:thin:@127.0.0.1:1521:XE" /><!-- Endereço do banco de dados -->

      <property name="hibernate.dialect"
value="org.hibernate.dialect.Oracle10gDialect" /><!-- Configura o dialeto que o Hibernate utilizará
para a montagem dos comandos SQL -->
      <property name="hibernate.max_fetch_depth" value="3" />

      <property name="hibernate.hbm2ddl.auto" value="create-
drop" />

      <property name="hibernate.show_sql" value="true" />
    </properties>
  </persistence-unit>
</persistence>

```

ANEXO NA – Código da tabela Jogadores

```
CREATE TABLE "JOGADORES"  
(  
    "JOG_CODIGO" NUMBER(4,0) NOT NULL ENABLE,  
    "JOG_NOME" VARCHAR2(30) NOT NULL ENABLE,  
    "JOG_APELIDO" VARCHAR2(3) NOT NULL ENABLE,  
    "JOG_DT_CRIACAO" VARCHAR2(10) NOT NULL ENABLE,  
    PRIMARY KEY ("JOG_CODIGO") ENABLE  
);
```

ANEXO AO – Código da tabela Recordes

```
CREATE TABLE "RECORDES"  
(  
    "REC_CODIGO" NUMBER(10,0) NOT NULL ENABLE,  
    "REC_DT_CRIACAO" DATE,  
    "REC_QUANTIDADE" NUMBER(10,0) NOT NULL ENABLE,  
    PRIMARY KEY ("REC_CODIGO") ENABLE  
);
```

ANEXO AP – Código da sequence Inc_Jog

```
CREATE SEQUENCE INC_JOG  
START WITH 1  
INCREMENT BY 1  
MINVALUE 1  
MAXVALUE 999;
```


ANEXO AQ – Código da sequence Inc_Rec

```
CREATE SEQUENCE INC_REC  
START WITH 1  
INCREMENT BY 1  
MINVALUE 1  
MAXVALUE 999;
```