

**UNIVERSIDADE PAULISTA
CIÊNCIA DA COMPUTAÇÃO**

**DANILO DE SALES SANTANA – C3228B1
DIOGO RAMOS LOPES DA SILVEIRA – C173398
RICARDO FERNANDES SOUTO – C13CHI5
WANDERSON MARTINS OLIVEIRA – C202754
YURY RODRIGUES ANUNCIAÇÃO – C203360**

**DESENVOLVIMENTO DE UM JOGO COM UTILIZAÇÃO DE INTERFACE
GRÁFICA**

**SÃO PAULO
2015**

SUMÁRIO

1 OBJETIVOS.....	2
1.1 Geral.....	2
1.2 Específicos.....	2
2 INTRODUÇÃO.....	3
3 REGRAS E FUNCIONAMENTO DO JOGO.....	5
3.1 Lixos e lixeiras.....	5
3.2 Jogador – Gari.....	6
3.3 Pontuação e vida.....	6
3.4 Movimentação dos personagens.....	7
3.5 Fases.....	7
3.6 Teclas.....	8
3.7 Mensagens no jogo.....	9
3.8 Menus.....	9
4 PLANO DE DESENVOLVIMENTO DO JOGO.....	10
4.1 Ambiente de desenvolvimento.....	10
4.2 Divisão da equipe.....	10
4.3 Escolha das imagens.....	11
4.4 Fonte.....	12
4.5 Nome e logomarca do jogo.....	12
4.6 Interface gráfica.....	13
4.7 Padrão de projeto.....	13
4.8 Armazenamento da pontuação dos jogadores.....	14
5 PROJETO DO PROGRAMA.....	16
5.1 View.....	16
5.1.1 GUI.....	16
5.1.2 Fonte.....	17
5.1.3 SobreJogo e Ajuda.....	17
5.1.4 RankPontuacao.....	18
5.1.5 SalvaPontuacao.....	18
5.1.6 JTextFieldLimite.....	18
5.1.7 Jogo.....	19
5.2 Control.....	20

5.2.1 RankPontuacaoCtrl.....	20
5.2.2 SalvaPontuacaoCtrl.....	20
5.2.3 RodarJogoCtrl.....	21
5.2.4 InputCtrl.....	21
5.2.5 JogoCtrl.....	22
5.3 Model.....	23
5.4 Dados.....	23
6 RELATÓRIO COM AS LINHAS DE CÓDIGO DO PROGRAMA.....	24
7 APRESENTAÇÃO DO JOGO EM FUNCIONAMENTO EM UM COMPUTADOR..	34
8 INTERDISCIPLINARIDADE.....	53
REFERÊNCIAS.....	54

1 OBJETIVOS

1.1 Geral

Desenvolver um jogo que contenha interface gráfica, utilizando a linguagem de programação Java. Tendo como tema a educação ambiental focada na vida na grande metrópole.

Buscando contribuir para a educação da população brasileira das grandes cidades através de um jogo divertido e envolvente.

1.2 Específicos

São objetivos específicos deste trabalho:

- Definir o tipo de jogo que será criado;
- Elaborar as regras do jogo;
- Definir a estrutura do projeto;
- Criar a interface gráfica;
- Implementar o jogo utilizando a linguagem de programação Java.

2 INTRODUÇÃO

Os problemas ambientais nas grandes metrópoles estão relacionados com o avanço das indústrias, que atraiu um grande número de pessoas para o centro das cidades. O processo de urbanização aconteceu sem nenhum planejamento, causando uma falta de infraestrutura adequada. Esse crescimento desordenado trouxe consequências ruins para o meio ambiente. Abaixo segue alguns desses problemas:

Poluição do ar – as emissões de poluentes são realizadas pelas indústrias e automóveis, sendo responsáveis por doenças respiratórias.

Poluição dos rios – o esgoto não é tratado de forma correta, e acaba sendo despejado nos rios, tornando inevitável a poluição das águas.

Ilha de calor – é um fenômeno climático que acontece nas regiões urbanas onde existe uma grande concentração de prédios, asfalto e concreto, causando o aumento da temperatura.

Efeito estufa – aumento da temperatura no planeta devido gases poluentes emitidos pelas cidades.

Erosão – causada pela ocupação irregular em áreas de preservação ambiental, como encostas e margens de rios, além do excesso de peso das edificações.

Enchentes – a água da chuva não tendo para onde escoar provoca enchentes nas grandes cidades, causando grandes prejuízos.

Lixo – é um dos principais problemas. O grande número de pessoas nas cidades acaba produzindo toneladas de lixo diariamente.

Este fenômeno tende a aumentar. Conforme a população urbana cresce, a quantidade de lixo segue na mesma direção, tornando preocupante o futuro da humanidade.

O lixo é responsável por uma série de problemas, como exemplos podemos citar a liberação de gases que provocam o efeito estufa, a poluição das águas subterrâneas e dos rios, a proliferação de insetos e doenças, além da poluição visual(percebida mais facilmente).

Para facilitar o manuseamento do lixo, eles são classificados em 3 grandes categorias, sendo elas: orgânico, inorgânico e tóxico.

Orgânico – material de origem biológica, como restos de alimentos, bebidas,

plantas, animais mortos, etc.

Inorgânico – inclui todo material que não é biológico: plástico, papel, vidro, etc.

Tóxico – resíduos que provem de substâncias químicas prejudiciais aos seres humanos. Pilhas, baterias, agrotóxico, descartes hospitalares e industriais são exemplos de lixos tóxicos.

Neste cenário, a reciclagem é a forma mais eficiente de combater o excesso de lixo. E para isto, é preciso investir mais na educação da população, conscientizando sobre a gravidade deste assunto, mostrando o que está em jogo: o futuro do planeta.

Muitos produtos são descartados de forma irregular, sendo que é necessário fazer um processo seletivo dos mesmos, dando um fim adequando a cada tipo de material. O plástico e o vidro, por exemplo, demoram anos para se decomporem na natureza, mas, quando reciclado, pode ser reutilizado, diminuindo o impacto ambiental causado pelo seu uso. Outro exemplo, mais comum, é o das árvores, que poderiam ser poupadadas se todo papel usado no cotidiano fosse reciclado.

Quando se trata de educar uma população é preciso utilizar todos os meios disponíveis, para atingir e conscientizar com eficiência e qualidade o maior número de pessoas possíveis. A utilização de jogos educacionais para alcançar este objetivo tem demonstrado grandes resultados, principalmente com crianças, que tendem a ter mais interesse quando o aprendizado é uma brincadeira.

As crianças de hoje crescem imersas em tecnologia, rodeadas de aplicativos e jogos, portanto, saber utilizar tais meios, com criatividade e de um jeito divertido, é uma maneira extremamente eficiente de atingir este público-alvo.

“Eduquem as crianças, para que não seja necessário punir os adultos”, a célebre frase do filósofo e matemático grego Pitágoras, se aplica muito bem a este cenário. Educando uma criança teremos um cidadão consciente e participativo, contribuindo para melhorar o meio ambiente das grandes cidades.

3 REGRAS E FUNCIONAMENTO DO JOGO

Primeiramente, é preciso definir o tipo de jogo que será desenvolvido. Basicamente, o jogo lembra, vagamente, o antigo clássico Space Invaders. Será, portanto, um jogo de atirar nos inimigos vindos do “espaço”.

Definido o tipo de jogo, é necessário desenvolver a história e os personagens por trás dele. Neste jogo, existirá apenas um personagem, o jogador. Ele será um gari das grandes cidades, encarregado de coletar o lixo jogado na rua pela população, tentando impedir que o bueiro entupa e provoque enchentes quando chover.

Os lixos serão os inimigos, e o gari o herói do jogo.

3.1 Lixos e lixeiras

Figura 1 – Tipos de lixo e lixeira

Tipo	Lixo	Lixeira
Papel		
Plástico		
Vidro		
Metal		
Orgânico		

Fonte: Elaborada pelos autores

Os lixos atirados pela população cairão do céu e o gari ficará na parte de

baixo da tela utilizando diversas lixeiras e tentando coletar o maior número possível deles. O lixo deverá ser coletado com a lixeira correspondente ao tipo do material de que ele é feito.

Cada inimigo representa um lixo da vida real, e como tal, é feito de um tipo específico de material. Sendo assim, deverá ser atingido pela lixeira do mesmo tipo. Este jogo é de coleta seletiva, portanto, é preciso seguir as regras da separação dos lixos.

Enquanto que no mundo real o lixo que é atirado na lixeira, no jogo é o contrário. Mas, o conceito por trás da escolha da lixeira é o mesmo. Existem 5 tipos de lixo e suas lixeiras correspondentes, conforme visto na Figura 1.

3.2 Jogador – Gari

O gari tem características e ações básicas. Por exemplo, enquanto que em muitos jogos deste tipo a vida do jogador está relacionada ao personagem controlado por ele, neste jogo, o gari não tem um contador de vidas.

O jogador consegue apenas movimentar o personagem e trocar as lixeiras que serão atiradas por ele. Isto torna a mecânica do jogo muito simples e intuitiva.

Na Figura 2 é mostrado o gari segurando os tipos de lixeira disponíveis.

Figura 2 – Gari com os 5 tipos de lixeira



Fonte: Elaborada pelos autores

3.3 Pontuação e vida

O jogador, gari, atira uma lixeira e quando a lixeira atinge um lixo, caso o lixo e a lixeira sejam do mesmo tipo, o jogador ganha 10 pontos, caso sejam de tipos diferentes ele perde 10 pontos. Se a lixeira não acertar nenhum lixo, quando ela ultrapassar o topo da tela ela desaparece e não ocorre nenhuma alteração na pontuação do jogador.

Quando um inimigo atinge o chão, ele some da tela e o jogador perda 10

pontos, além da vida do bueiro ser diminuída em 10%(no início do jogo ela está com 100%).

O conceito de vida do jogo não é relativo ao gari e sim a um bueiro invisível. Este bueiro representa um bueiro da vida real. E todo lixo que atingi o chão escorrega para o bueiro, entupindo ele, e consequentemente diminuindo sua vida. No mundo real, quando chove e um bueiro está entupido, ocorre enchentes. O jogo segue um conceito parecido, quando o bueiro entope(sua vida chega a 0%), o jogador perde o jogo, simbolizando que ele não conseguiu evitar o desastre.

Existe um sistema de classificação dos jogadores, um top-100 dos que mais pontuaram. Mas, o número máximo de pontos que um jogador pode ter será fixo. Cada fase sempre terá a mesma quantidade de inimigos e todos valendo 10 pontos. Portanto, poderá existir mais de um jogador que alcançou a pontuação máxima.

3.4 Movimentação dos personagens

O gari somente se movimenta na horizontal, no limite da tela, nunca ultrapassando. Os inimigos, lixos, fazem o movimento no sentido vertical, de cima para baixo. O movimento vertical não existe para o gari e o movimento horizontal não existe para os inimigos.

Embora em muitos jogos do gênero os inimigos caiam realizando movimentos, como o zigue-zague, neste jogo, eles não tem nenhum movimento além da queda em linha reta.

3.5 Fases

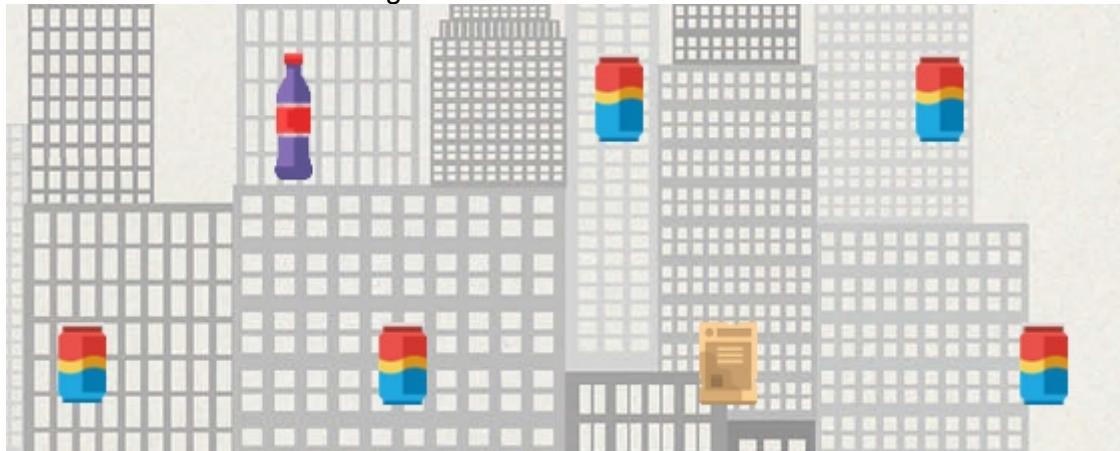
Os inimigos virão do topo da tela em fileira. Na primeira fase será 1 inimigo por fileira. Na segunda fase 2 inimigos. E assim sucessivamente, até atingir a 5º fase, onde o limite será de 5 inimigos por fileira. Portanto, o jogo terá 5 fases e no máximo 5 inimigos por fileira.

Os inimigos serão gerados de forma aleatória. Não existirá controle sobre quais tipos de inimigos uma fileira terá. Isto torna o jogo mais imprevisível e envolvente.

Cada fase terá 20 fileiras de inimigos, sendo eles intercaláveis. Para explicar melhor, peguemos como exemplo a quinta fase. A tela do jogo tem capacidade para

mostrar 10 inimigos por fileira. A primeira fileira terá 5 inimigos, restando portanto 5 espaços vagos. A próxima fileira irá preencher 4 destes espaços com inimigos. Desta forma mascara, um pouco, o fato dos inimigos estarem caindo em fileiras verticais. Veja a Figura 3 para entender melhor o conceito.

Figura 3 – Fileiras intercaláveis



Fonte: Elaborada pelos autores

Ainda com relação as fases, cada uma delas terá uma imagem de fundo diferente, além da velocidade dos lixos ser menor do que a da fase anterior. Isto se deve ao fato de que aumentando a quantidade de inimigos por fileira é preciso diminuir a velocidade dos inimigos, senão, fica impossível vencer o jogo.

Existe uma diferença de velocidade entre os lixos e o gari. Os lixos serão sempre mais lentos do que o jogador. Isto permite que o gari percorra a tela de uma ponta a outra capturando os inimigos antes que eles atinjam o chão. Se não tiver este mecanismo, o jogador não conseguirá atingir todos os inimigos de uma fileira, pois sua velocidade será insuficiente.

3.6 Teclas

As teclas utilizadas no jogo são as *setas direita e esquerda*, as teclas *A-S-D-F-G* e o *Espaço*. As setas movimentam o jogador pela horizontal, as teclas alfabéticas trocam a lixeira do gari e o *Espaço* atira a lixeira no inimigo.

Além das teclas de ação do jogo, existem mais duas teclas que o jogador pode utilizar. A tecla *P*, para pausar o jogo, e a tecla *ESC* para sair do jogo(fechar a janela).

3.7 Mensagens no jogo

Quando o jogador perde ou ganha o jogo, é exibida uma tela perguntando se ele deseja iniciar um novo jogo ou salvar sua pontuação. Caso escolha um novo jogo, ele é imediatamente iniciado.

Se preferir salvar a pontuação, o jogo verifica se sua pontuação ficou no top-100. Caso tenha ficado, pede o nome do jogador e salva sua pontuação. Se tiver pontuação abaixo do top-100, é mostrada esta informação e o botão de salvar pontos é desabilitado.

3.8 Menus

O jogo conta com uma barra de menus com algumas opções. As mais relevantes são a de ajuda e a de exibir a classificação dos jogadores. A opção de ajuda exibe uma tela com as regras do jogo e as teclas que podem ser utilizadas pelo jogador. Já a opção de classificação mostra os 100 jogadores que mais pontuaram.

4 PLANO DE DESENVOLVIMENTO DO JOGO

4.1 Ambiente de desenvolvimento

O jogo foi desenvolvido com o uso da linguagem de programação Java e o paradigma de Orientação a Objetos. A IDE utilizada foi o **Eclipse**(na versão Mars), com a **JRE versão 1.8**.

O GIT foi empregado no controle das versões do jogo. Para isto foi adicionado o *plugin EGit* no Eclipse. Permitindo maior segurança, velocidade e produtividade no desenvolvimento do projeto.

Outro *plugin* utilizado no Eclipse foi o **ObjectAid**. Ele permite a geração de arquivos UML e de diagramas do projeto. Todos os arquivos destes tipos, utilizados neste trabalho, foram criados com o uso deste *plugin*.

Para o tratamento das imagens do jogo foi utilizado o **GIMP**. Um programa *open source* e multiplataforma, que conta com a grande maioria das funcionalidades do Photoshop, sendo uma de suas vantagens ser gratuito.

Figura 4 – Logos dos programas e *plugins* utilizados



Fonte: Das respectivas empresas e projetos.

4.2 Divisão da equipe

Com o gênero, história e conceitos do jogo previamente decididos, a equipe

de desenvolvimento foi dividida em duas. A primeira ficando responsável pela parte gráfico e a última pela codificação da lógica do jogo.

A parte gráfica diz respeito as imagens e todos os elementos visíveis do jogo. Isto inclui também os menus e telas de mensagens exibidas para o usuário. O conceito visual do jogo ficou sob a responsabilidade da equipe cujos membros já tinham conhecimentos prévios sobre a utilização de programas de tratamento de imagens.

A segunda equipe, por sua vez, recebeu a tarefa de desenvolver, desde o princípio, a lógica por trás do jogo. Todo o tratamento de ações, eventos, movimentação, sistemas de colisões, pontuações, e demais regras pertinentes a tornar o jogo jogável, ficou com esta equipe.

Após a parte gráfico estar concluída, a equipe responsável por esta tarefa se uniu a segunda equipe, ajudando no desenvolvimento dos módulos do jogo.

4.3 Escolha das imagens

Com relação as lixeiras utilizadas no jogo, foi preciso escolher quais tipos seriam utilizadas. Existem pelo menos 10 modelos, cada uma específica para um tipo de lixo: papel(e papelão), plástico, vidro, metal, resíduos radioativos, resíduos orgânicos, resíduos não recicláveis, madeira, resíduos perigosos e resíduos dos serviços de saúde.

Para manter o conceito de coleta seletiva o mais próximo da realidade, foi utilizado no jogo somente os principais tipos de lixeiras disponibilizadas em locais públicos: papel, plástico, vidro, metal e orgânica.

Com esta definição estabelecida, a próxima etapa era escolher as imagens para serem utilizadas no jogo. As lixeiras seriam das cores de seus tipos, o gari vestiria roupas laranjas(a cor mais utilizada nos uniformes destes profissionais), e os lixos seriam de tipos e cores diferentes, mas, com conceitos visuais próximos, para tornar mais harmônica a identidade visual dos elementos. Por último, o plano de fundo das fases, que deveria remeter as grandes cidades.

Ainda com relação ao gari, ele seguraria a lixeira que seria atirada, e portanto, deveriam ser feitas duas imagens para cada tipo de lixeira. Uma imagem seria a da lixeira atirada(sozinha), e a outra estaria unida a imagem do jogador.

Para manter o jogo simples, foi preferido não implementar imagens que

representassem o gari caminhando. Sendo preciso apenas uma imagem estática do jogador.

4.4 Fonte

Para criar uma melhor identidade visual, foi escolhida a fonte **Ubuntu Regular**(Figura 5) para ser utilizada nos textos exibidos pelo jogo.

Esta fonte foi desenvolvida pela Canonical, e é utilizada na distribuição Linux Ubuntu, também desenvolvida e mantida pela mesma empresa. A licença de uso da fonte permite sua livre utilização, seja em documentos ou em programas.

Figura 5 – Texto formatado utilizando a fonte Ubuntu Regular

Ubuntu Regular
 abcdefghijklmnopqrstuvwxyz
 ABCDEFGHIJKLMNOPQRSTUVWXYZ
 0123456789.:;(*!?)

Fonte: Elaborada pelos autores

4.5 Nome e logomarca do jogo

O nome escolhido para o jogo foi “ataque do lixo”, traduzido para o inglês. Para a fonte do texto foi selecionada a mesma fonte da interface gráfica, a Ubuntu.

Um lixo atingindo uma das letras foi a forma encontrada de deixar a logomarca mais bonita, dinâmica, energética, e tirando o ar de monotonia com que ficaria se fosse utilizado somente letras.

Na Figura 6 é mostrada a logomarca finalizada, implementando todas as características que foram previamente definidas.

Figura 6 – Logomarca do jogo



Fonte: Elaborada pelos autores

4.6 Interface gráfica

Para desenvolver a interface gráfica do jogo será utilizada a biblioteca gráfica Swing. Ela está disponível por padrão em todas as JRE, a partir do Java 1.2. Sua principal vantagem reside no fato de ser independente do sistema operacional. Ou seja, uma vez desenvolvida a GUI, ela será a mesma, com os mesmos componentes, cores e tamanhos, em qualquer OS e em qualquer versão do mesmo, seja no Windows, MacOs ou Linux.

A biblioteca Swing deverá ser utilizada apenas para a barra de menus e as caixas de diálogo da janela. Textos exibidos dentro do próprio jogo não utilizarão esta biblioteca.

4.7 Padrão de projeto

Neste jogo será utilizado o padrão de projeto MVC(Model-View-Controller). Uma de suas principais vantagem reside no fato de dividir a estrutura da aplicação em 3 camadas, facilitando alterações no código, atualizações e correções. Além disto, pelo fato de estar dividido e com regras bem definidas, pode-se ter uma grande equipe trabalhando simultaneamente no mesmo projeto, sem grandes dificuldades.

Isto facilita e aumenta a produtividade no desenvolvimento. Garantindo também maior segurança, pois quando se altera uma camada, você tem certeza que não afetará diretamente a outra.

O ponto central do MVC é o relacionamento que existe entre as partes do projeto. Definindo quem, quando e se uma das partes pode comunicar-se com outra. Sendo assim, é feita a divisão em 3 camadas e aplicada as regras do MVC, para definir como será a interação entre estas partes.

Existe uma pequena diferença nesta implementação do MVC. Normalmente, a View não se comunica com o Model, existe uma camada intermediária entre as duas. A View se comunica com o Controller e este se comunica com o Model. Neste projeto, a View terá uma comunicação direta com o Model.

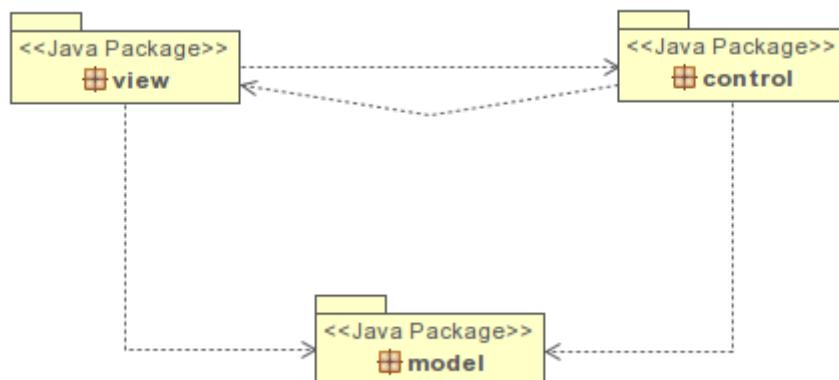
Qualquer ação do usuário será tratada pelo Controller. Além disto, é o Controller que notificará o Model para executar alterações em seus dados. A View nunca pedirá para o Model alterar seus dados, e também nunca irá alterá-los

diretamente. A View somente pegará os dados do Model e exibirá.

Isto ocorre pelo fato de no jogo a camada View ser muito dependente dos dados do Model. A todo instante, centenas de vezes por segundo, a View utilizará dados do Model para desenhar **a tela inteira**(não somente algumas partes, como em formulários). Acessar diretamente os dados, portanto, reduzirá a duplicidade de dados nas três camadas e aumentará a velocidade de comunicação entre elas.

Este tipo de abordagem não é inteiramente nova, sendo utilizada desde o início do MVC. No entanto, com o passar do tempo, foi sendo adotado o conceito de que a View não tem ligação direta com o Model. Portanto, estaremos apenas utilizando um padrão mais próximo do original.

Figura 7 – MVC utilizado no jogo



Fonte: Elaborada pelos autores

4.8 Armazenamento da pontuação dos jogadores

Como o jogo terá um sistema de classificação da pontuação dos jogadores, é preciso salvar estas pontuações de alguma forma. Existem dois principais métodos para isto.

O primeiro utiliza um banco de dados para o armazenamento dos dados. Esta maneira é a menos indicada para o tipo de jogo desenvolvido neste projeto. Isto se deve a um grande problema, neste cenário, causado por este método: o banco de dados deve ser instalado e mantido ativo em algum servidor.

Se o servidor for a própria máquina do jogador, será preciso instalar um programa adicional no computador do usuário. Tornando o pacote do jogo mais pesado, consumindo recursos extras do computador, e instalando um programa SGBD que dificilmente o jogador utilizará para outra atividade. Ou seja,

desperdiçaremos grandes recursos em um simples jogo, sem existir necessidade para isto.

A segunda maneira de salvar os dados da classificação é utilizando um arquivo de texto. Uma maneira simples, eficiente, de fácil implementação, que não necessita da instalação de programas adicionais, e que não desperdiça recursos extras.

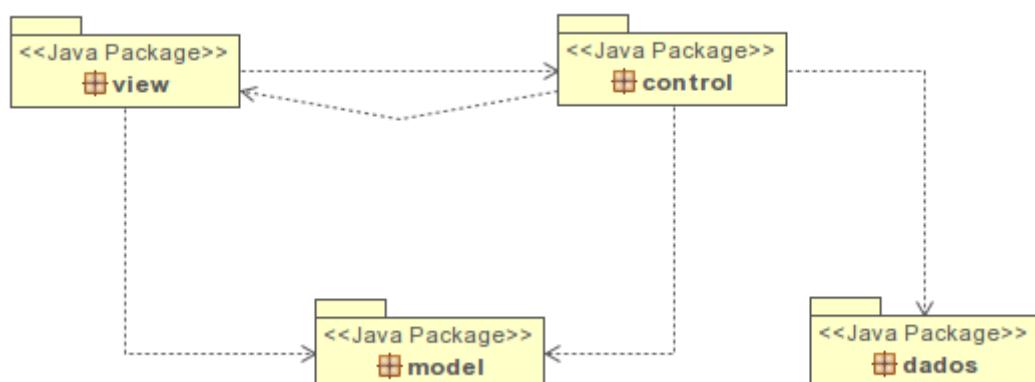
Este arquivo de texto guardará apenas a pontuação dos jogos ocorridos no computador em que o programa está instalado. Portanto, a classificação será local, não será uma classificação global, de todos os jogadores do “mundo”.

Pode, em um primeiro momento, parecer uma falha do programa, mas, o jogo não tem como foco ser um sistema de competição global. Seu objetivo é apenas divertir o jogador e fazê-lo tentar quebrar o seu próprio recorde. Não é uma modalidade nova, muitos jogos de smartphone e computador utilizam tal recurso. Um exemplo é o SuperTuxKart, que embora seja um grande jogo, utiliza tal recurso.

O arquivo de texto será armazenado em uma pasta ao lado do executável do jogo. Poderia ser feito de outra forma, escondendo este arquivo na pasta de configurações do usuário do computador, e junto com outras técnicas, impedir alterações indevidas nele. Mas, foi preferido manter o armazenamento da maneira mais simples. Caso o jogo fosse efetivamente ser distribuído em larga escala, seria preciso adicionar esta camada extra de segurança.

No interior do arquivo será salvo apenas o nome do jogador e sua pontuação. Depois de ler os dados, o programa deverá separar os nomes dos jogadores de seus respectivos pontos, e em seguida ordenar estes dados. Promovendo desta forma, uma maior segurança contra alterações manuais na ordem da classificação.

Figura 8 – MVC com a camada “dados”



Fonte: Elaborada pelos autores

5 PROJETO DO PROGRAMA

Com a estrutura do programa definida, que terá como base o MVC, com o projeto dividido em 3 camadas principais e uma camada extra, para o armazenamento e recuperação dos dados, é necessário desenvolvê-las, para efetivamente criar o jogo.

A primeira classe utilizada estará fora destas camadas, e será a classe principal do jogo. Sua única função será iniciar a interface gráfica.

Programa 1 – Classe Principal

```
import view.GUI;
public class Principal {
    public static void main(String args[]){
        new GUI();
    }
}
```

Fonte: Elaborado pelos autores

5.1 View

Este pacote contém as classes View do MVC e os demais arquivos relacionados a interface gráfica.

5.1.1 GUI

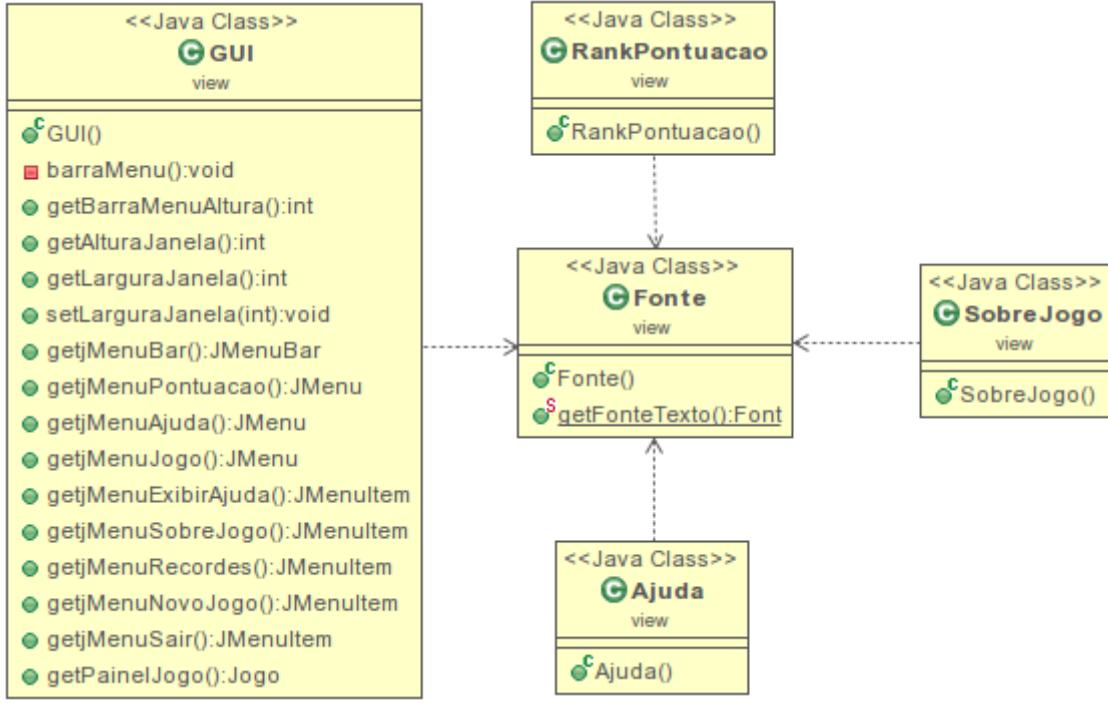
Esta é a principal classe do pacote View, é ela que exibirá a janela do jogo. Será chamada pela classe **Principal** assim que o programa for executado. A Figura 9 mostra os métodos desta classe. Os atributos foram ocultados, pois não são relevantes para esta explicação.

Seu método privado **barraMenu** constrói e adiciona uma barra de menu na janela, contendo todas as opções disponíveis para o usuário. Os métodos que retornam largura e altura, como **getAlturaJanela** e **getLarguraJanela**, serão invocados pelas classes que precisam saber estas informações para, por exemplo, ajustar o tamanho da tela e adicionar corretamente os elementos do jogo na janela.

O método **getPainelJogo** retorna o painel do jogo, sendo utilizado principalmente para alterar e exibir mensagens dentro do painel. Além de ser utilizado para acessar o controlador deste componente.

Os métodos que retornam o Menu do jogo(exemplo, `getjMenuPontuacao`) servem para o controlador da GUI identificar e tratar os cliques do usuário na barra de menus.

Figura 9 – Parte das classes do pacote View



Fonte: Elaborada pelos autores

5.1.2 Fonte

Esta classe tem como única função importar o arquivo da fonte utilizada no jogo, neste caso a fonte Ubuntu. Tem somente um método, e serve para retornar a referência para o arquivo importado.

Todos os textos da interface gráfica e mensagens do jogo, irão ser exibidos na fonte especificada, e portanto, utilizarão a classe **Fonte**. Como ela manterá um único dado imutável, que será o mesmo para todas as partes do jogo, seus métodos serão estáticos. Não precisando ser instanciado nenhum objeto a partir dela.

5.1.3 SobreJogo e Ajuda

As classes **SobreJogo** e **Ajuda** exibirão apenas textos estáticos. A primeira com as informações sobre o jogo, sua versão e seus desenvolvedores. A segunda mostrará o objetivo do jogo e quais as teclas que o usuário pode utilizar.

Por este motivo, elas não terão uma classe de controle.

5.1.4 RankPontuacao

Sua função será mostrar a classificação das pontuações dos jogadores. O limite de jogadores pertencentes a classificação será de 100. Portanto, será um top 100 dos melhores jogadores.

Esta classe será do tipo *JScrollPane* e utilizará uma *JTable* para exibir os dados. Permitindo que o usuário role o ranking e veja todos os jogadores classificados.

Para gerar o ranking será utilizada a classe *RankPontuacaoCtrl*. Ela passará para a classe do pacote View os dados que deverão ser exibidos e em qual ordem.

5.1.5 SalvaPontuacao

Na Figura 10 são mostradas as últimas três classes do pacote View. Embora não esteja demonstrado na figura, estas classes, com exceção de *JtextFieldLimite*, assim como todas as outras, utilizarão a classe *Fonte* para definir a fonte dos textos que serão exibidos.

A classe *SalvaPontuacao* mostrará uma tela onde o usuário poderá salvar sua pontuação. Ela exibirá apenas um campo editável, reservado para o usuário digitar seu nome. A classe *SalvaPontuacaoCtrl* será sua controladora, que tratará os eventos do campo de texto, a validação do nome do usuário e a verificação se é possível salvar a pontuação.

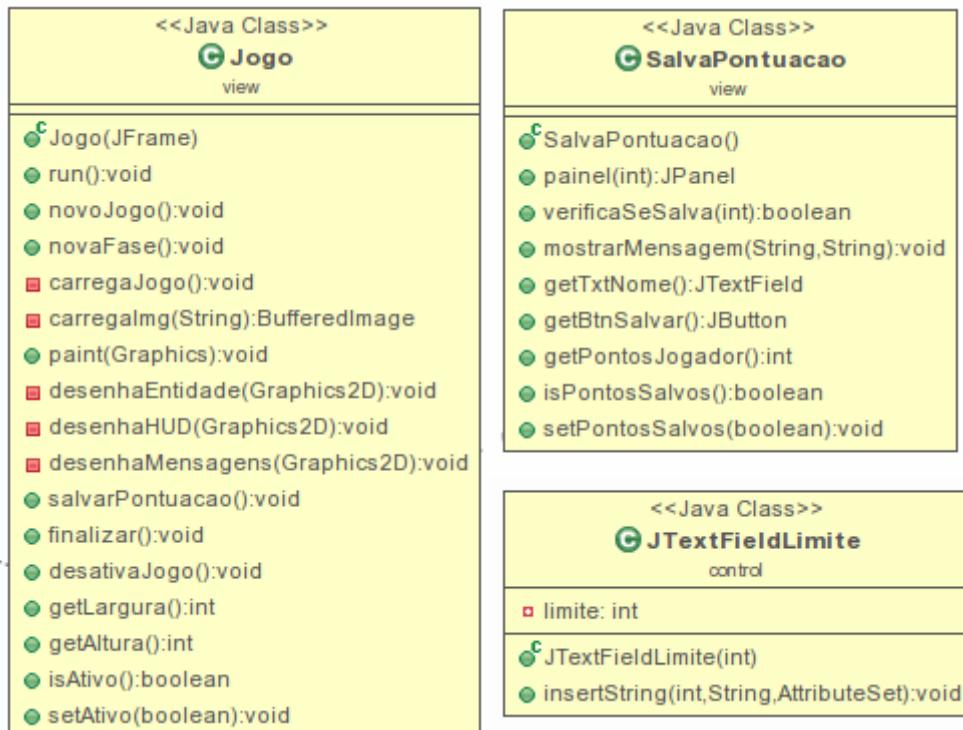
O método *isPontosSalvos* retorna a indicação se o jogador salvou seus pontos ou não. Isto se deve ao fato de esta janela ser exibida apenas quando o jogador ganha ou perde o jogo, precisando ser desabilitado o botão que abre esta janela quando o jogador salva sua pontuação.

5.1.6 JtextFieldLimite

Esta classe é utilizada para limitar a quantidade de caracteres que podem ser digitados em um campo de texto. Sendo instanciada na classe *SalvaPontuacao*, para definir a número máximo de letras que o jogador pode ter no seu nome.

Ela estende a classe **PlainDocument** e para todo caractere digitado pelo usuário ela verifica se já atingiu a quantidade máxima definida, se já atingiu ela impedi sua adição no campo de texto, caso contrário, adiciona o caractere digitado.

Figura 10 – Parte das classes do pacote View



Fonte: Elaborada pelos autores

5.1.7 Jogo

Para finalizar o pacote View, temos a classe **Jogo**. Ela é o painel onde será exibido jogo. Gari, lixeiras, lixos, fases, planos de fundo, em fim, todos os gráficos do jogo serão desenhados neste painel, ele é um dos núcleos do jogo.

Os métodos **get's** e **set's** foram ocultados, pois, neste momento, não tem necessidade de explicá-los. Os métodos **novoJogo**, **novaFase**, **carregaJogo** e **carregalmg**, instanciam e importam os objetos e imagens necessários para exibir o jogo, além de definir as suas configurações. O método **run**, instancia o objeto controlador do loop do jogo, iniciando efetivamente o jogo para o jogador jogar.

O método **salvarPontuacao**, exibe a janela de salvamento de pontos(da classe **SalvaPontuacao**) caso o jogador tenha ganho ou perdido o jogo.

Esta classe terá 3 classes controladoras, **JogoCtrl** definirá como os dados do jogo devem ser alterados e exibidos pela View, **InputCtrl** tratará os eventos do

teclado e do mouse, e por último, **RodarJogoCtrl** que controlará o loop do jogo.

5.2 Control

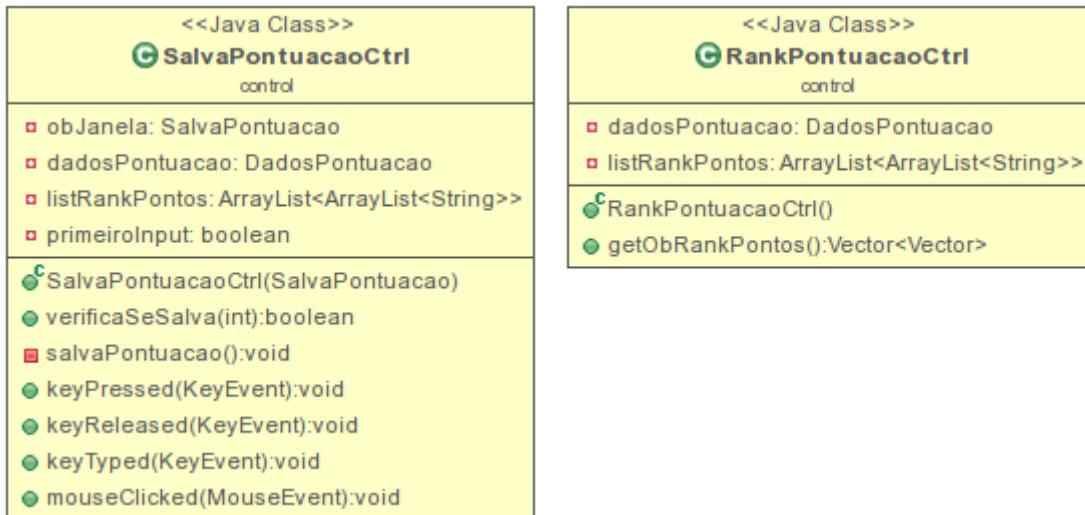
A segunda camada do jogo é a Control, ela contém todas as classes de controle, cuja principal função é tratar os eventos do jogo, sejam eles da interface, teclado ou mouse. Além disto, também tratam as entradas de dados do usuário.

5.2.1 RankPontuacaoCtrl

A classe controladora mais simples. Ela instancia um objeto do tipo **DadosPontuacao**, que é responsável por retornar os dados das pontuações salvas, ordenados do menor para o maior. Em seguida esta classe controladora formata os dados para serem exibidos pela classe **RankPontuacao** do pacote View.

Na Figura 11 são mostradas as duas primeiras classes do pacote Control.

Figura 11 – Parte das classes do pacote Control



Fonte: Elaborada pelos autores

5.2.2 SalvaPontuacaoCtrl

Esta classe instancia um objeto do tipo **DadosPontuacao**, para pegar o ranking de pontos do jogo.

O método **verificaSeSalva**, compara a pontuação do jogador com a classificação dos pontos, se a pontuação estiver dentro do top 100, indica que a

View pode exibir a janela para o jogador digitar seu nome. Caso contrário, indica que deve ser exibida uma janela informando que a pontuação do jogador é insuficiente para entrar no ranking.

Já o método **salvaPontuacao**, utiliza o objeto **DadosPontuacao** para substituir a menor pontuação do ranking pela pontuação do jogador, se ele estiver dentro do top-100.

5.2.3 RodarJogoCtrl

A classe **RodarJogoCtrl** controlará o loop do jogo. Nela está definido o limite de vezes que o loop deve rodar por segundo. Desta forma a velocidade do jogo se mantém constante não importando a configuração do computador.

Se o computador for muito lento, o loop rodará menos vezes por segundo do que a quantidade definida. Neste caso, esta classe atualizará os dados do jogo e pulará algumas atualizações da View. Aumentando a sensação de velocidade do jogo.

Em caso do computador ser mais rápido do que o ideal, o jogo irá “dormir” por alguns milissegundos, para diminuir sua velocidade e ficar no limite definido.

Enquanto o jogo estiver ativo, o loop atualizará os dados do jogo e a interface gráfica, centenas de vezes por segundo!

5.2.4 InputCtrl

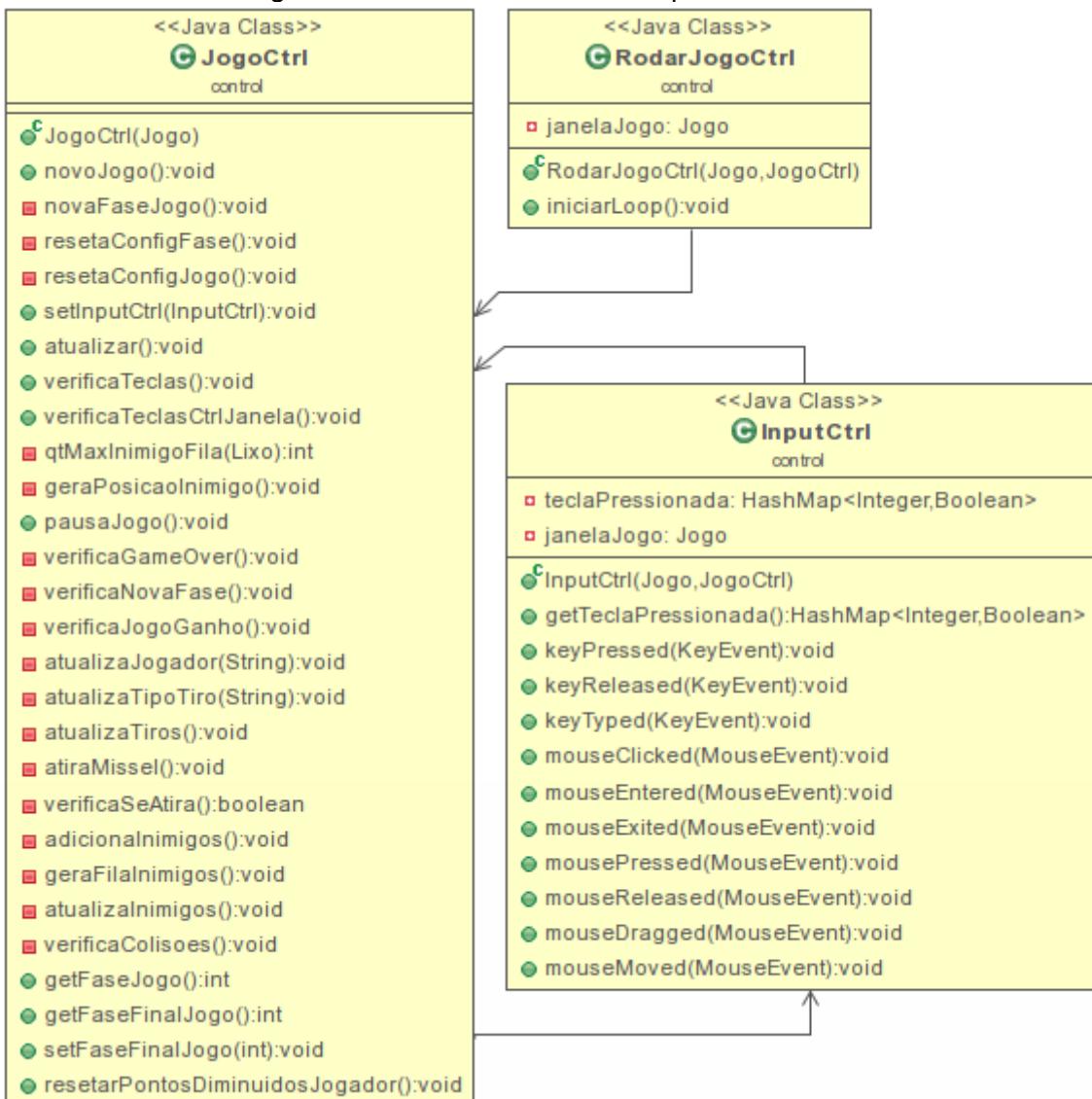
Esta classe é responsável por armazenar os eventos do teclado e do mouse. Os eventos do teclado, dentro do painel do jogo, são tratados posteriormente pela classe **JogoCtrl**. Já os eventos do mouse são tratados diretamente nesta classe.

Os eventos do último tipo, são relacionados aos botões de “Novo Jogo” e “Salvar Pontuação” que são exibidos dentro do painel do jogo(através da biblioteca Graphics2D) assim que o jogador perde ou ganha o jogo.

Quando o jogador posiciona o mouse dentro das áreas dos retângulos dos botões, seu ponteiro é mudado para o ponteiro do tipo clique. Quando os botões são clicados os seus respectivos métodos são acionados.

Utilizar este tipo de botão permite uma maior personalização de sua aparência se comparado a um simples JButton.

Figura 12 – Parte das classes do pacote Control



Fonte: Elaborada pelos autores

5.2.5 JogoCtrl

Na Figura 12 são mostradas as 3 últimas classes do pacote Control. Os atributos da classe **JogoCtrl** foram ocultados, pois são irrelevantes para a explicação.

Esta classe é o coração do jogo. A lógica do fluxo de dados, tratamento de eventos e exibição das Views, quando o jogo está ativo, se encontra nesta classe. Sua principal responsabilidade é transformar os eventos do teclado em comandos do jogo e acionando-os.

Nela também estão inclusos os métodos de construção das fases do jogo, os

de adição e remoção de inimigos, detecção de colisão, movimentação dos lixos e do jogador, sistema de tiros das lixeiras, aumento ou diminuição da pontuação do jogador, a definição se o jogo deve continuar e se o jogador ganhou ou perdeu o jogo.

5.3 Model

Dentro deste pacote se encontram as classes com os dados dos modelos utilizados no jogo. Existem duas classes abstratas. A primeira é a **Entidade**, que é estendida por todos os elementos visíveis do jogo(**Jogador**(gari), **Lixo** e **Tiro**(lixeiras atiradas)), e contém os atributos e métodos que eles devem possuir em comum, como por exemplo, as coordenadas na tela e a respectiva imagem do elemento.

A outra classe abstrata é a **TipoMaterial**, que contém métodos estáticos para definir o tipo dos lixos exibidos na tela e os tipos de lixeiras disponíveis, além, é claro, do tipo da lixeira selecionada pelo jogador. Sem esta classe não seria possível o sistema de colisão, que somente pontua o jogador se o tipo de lixeira atirada for do mesmo tipo que o lixo acertado.

5.4 Dados

Por último, temos o pacote Dados. Ele contém 3 classes. A primeira é **SalvaDados**, cuja função é salvar dados em um arquivo de texto especificado. Ela contém dois métodos de salvamento, o primeiro apaga todos os dados contidos no arquivo antes de escrever nele. O segundo método somente adiciona os dados no final do arquivo.

A segunda classe é a **LerDados** que contém os métodos de leitura de dados de arquivos de textos. Existem dois métodos de leitura nesta classe, o primeiro lê todas as linhas e o segundo ler somente a quantidade de linhas especificadas por parâmetro. Além disto um terceiro método retorna os dados lidos em forma de vetor.

Deste pacote a classe mais importante é a **DadosPontuacao**, que é responsável por definir o local de salvamento das pontuações dos jogadores, ler e salvar estes dados(utilizando as classes **LerDados** e **SalvaDados**), além de preparar os dados para serem utilizados pelas demais partes do programa.

6 RELATÓRIO COM AS LINHAS DE CÓDIGO DO PROGRAMA

A seguir estão expostos os códigos fontes das principais classes do jogo. Foi preferido ocultar o código das demais classes para que não ultrapasse o limite de 10 páginas reservadas a este tópico. As linhas de importação de bibliotecas e classes(**import**), foram ocultadas pelo mesmo motivo.

Programa 2 – Classe GUI

```

package view;
public class GUI extends JFrame {
    private int larguraJanela = 800;
    private int alturaJanela = 600;
    private String tituloJogo = "TrashAttack";
    private GUICtrl guiCtrl;
    private Jogo painelJogo;
    private JMenuBar barraMenu, jMenuBar;
    private JMenu jMenuPontuacao, jMenuAjuda, jMenuJogo;
    private JMenuItem jMenuExibirAjuda, jMenuSobreJogo;
    private JMenuItem jMenuRecordes, jMenuNovoJogo, jMenuSair;
    public GUI(){
        guiCtrl = new GUICtrl(this);
        setTitle(tituloJogo);
        setSize(larguraJanela,alturaJanela);
        setLocationRelativeTo(null);
        setResizable(false);
        setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        barraMenu();
        setJMenuBar(jMenuBar);
        painelJogo = new Jogo(this);
        add(painelJogo);
        pack();
        setVisible(true);
        painelJogo.run();
    }
    private void barraMenu(){
        jMenuBar = new JMenuBar();
        jMenuJogo = new JMenu();
        jMenuPontuacao = new JMenu();
        jMenuAjuda = new JMenu();
        jMenuSobreJogo = new JMenuItem();
        jMenuExibirAjuda = new JMenuItem();
        jMenuRecordes = new JMenuItem();
        jMenuNovoJogo = new JMenuItem();
        jMenuSair = new JMenuItem();
        jMenuBar.add(jMenuJogo);
        jMenuBar.add(jMenuPontuacao);
        jMenuBar.add(jMenuAjuda);
        jMenuJogo.setText("Jogo");
        jMenuJogo.add(jMenuNovoJogo);
        jMenuJogo.add(jMenuSair);
        jMenuPontuacao.setText("Pontuação");
        jMenuPontuacao.add(jMenuRecordes);
        jMenuAjuda.setText("Ajuda");
        jMenuAjuda.add(jMenuExibirAjuda);
    }
}

```

```

jMenuAjuda.add(jMenuSobreJogo);
jMenuSobreJogo.setText("Sobre o Jogo");
jMenuSobreJogo.addActionListener(guiCtrl);
jMenuExibirAjuda.setText("Exibir Ajuda");
jMenuExibirAjuda.addActionListener(guiCtrl);
jMenuSair.setText("Sair");
jMenuSair.addActionListener(guiCtrl);
jMenuNovoJogo.setText("Novo Jogo");
jMenuNovoJogo.addActionListener(guiCtrl);
jMenuRecordes.setText("Recordes");
jMenuRecordes.addActionListener(guiCtrl);
jMenuBar.setFont(Fonte.getFonteTexto()));
}
public int getBarraMenuAltura(){ return barraMenu.getHeight(); }
public int getAlturaJanela(){ return alturaJanela; }
public int getLarguraJanela() { return larguraJanela; }
public void setLarguraJanela(int larguraJanela) {
    this.larguraJanela = larguraJanela; }
public JMenuBar getjMenuBar() { return jMenuBar; }
public JMenu getjMenuPontuacao() { return jMenuPontuacao; }
public JMenu getjMenuAjuda() { return jMenuAjuda; }
public JMenu getjMenuJogo() { return jMenuJogo; }
public JMenuItem getjMenuExibirAjuda() { return jMenuExibirAjuda; }
public JMenuItem getjMenuSobreJogo() { return jMenuSobreJogo; }
public JMenuItem getjMenuRecordes() { return jMenuRecordes; }
public JMenuItem getjMenuNovoJogo() { return jMenuNovoJogo; }
public JMenuItem getjMenuSair() { return jMenuSair; }
public Jogo getPainelJogo() { return painelJogo; }
}

```

Fonte: Elaborado pelos autores

Programa 3 – Classe Jogo

```

package view;
public class Jogo extends JPanel {
private JFrame janelaJogo;
private boolean ativo, pause, gameOver, jogoGanho, novaFase, pontosSalvos;
private RodarJogoCtrl rodarJogoCtrl;
private JogoCtrl jogoCtrl;
private InputCtrl inputCtrl;
private String imgFundoJogo = "cenario.jpg";
private Jogador obJogador;
private ArrayList<Lixo> obInimigos;
private ArrayList<Tiro> obTiros;
private HashMap<String, BufferedImage> imgFundoTela, imgJogador;
private HashMap<String, BufferedImage> imgInimigos, imgTiros;
private Rectangle frmRtgNovoJogo, frmRtgSalvarPontos;
private double tempoInicioMsgJogo = 0;
private boolean exibeMenosPontos;
private double tempoInicioHUDJogo = 0;
public Jogo(JFrame janelaJogo){
    this.janelaJogo = janelaJogo;
    Dimension tamanhoPainel = new Dimension(janelaJogo.getWidth(),
janelaJogo.getHeight()-janelaJogo.getJMenuBar().getHeight()-27);
    setSize(tamanhoPainel);
    setPreferredSize(tamanhoPainel);
    setFocusable(true);
    setDoubleBuffered(true);
}

```

```

setIgnoreRepaint(true);
jogoCtrl = new JogoCtrl(this);
inputCtrl = new InputCtrl(this, jogoCtrl);
addKeyListener(inputCtrl);
addMouseListener(inputCtrl);
addMouseMotionListener(inputCtrl);
jogoCtrl.setInputCtrl(inputCtrl);
novoJogo();
carregaJogo();
}
public void run(){
    rodarJogoCtrl = new RodarJogoCtrl(this, jogoCtrl);
    rodarJogoCtrl.iniciarLoop();
}
public void novoJogo(){
    obJogador      = new Jogador(getLargura(), getAltura());
    obInimigos     = new ArrayList<>();
    obTiros        = new ArrayList<>();
    ativo          = true;
    pause          = false;
    gameOver       = false;
    jogoGanho      = false;
    novaFase       = false;
    pontosSalvos   = false;
    exibeMenosPontos = false;
}
public void novaFase(){
    obInimigos = new ArrayList<>();
    obTiros   = new ArrayList<>();
}
private void carregaJogo(){
    imgFundoTela  = new HashMap<String, BufferedImage>();
    for(int i=1; i<=jogoCtrl.getFaseFinalJogo(); i++){
        imgFundoTela.put(Integer.toString(i),
carregaImg("cenario/cenario-"+i+".png"));
    }
    ArrayList<String> tiposMateriais = TipoMaterial.getTipos();
    imgJogador    = new HashMap<String, BufferedImage>();
    Jogador j;
    imgTiros      = new HashMap<String, BufferedImage>();
    Tiro t;
    imgInimigos   = new HashMap<String, BufferedImage>();
    Lixo l;
    for(int i=0; i < tiposMateriais.size(); i++){
        j = new Jogador(getLargura(), getAltura());
        j.setTipoTiro(tiposMateriais.get(i));
        imgJogador.put(j.getImg(), carregaImg(j.getImg()));
        t = new Tiro(0,0, tiposMateriais.get(i));
        imgTiros.put(t.getImg(), carregaImg(t.getImg()));
        l = new Lixo(tiposMateriais.get(i));
        imgInimigos.put(l.getImg(), carregaImg(l.getImg()));
    }
}
private BufferedImage carregaImg(String img){
    URL urlImg = Jogo.class.getResource("/recursos/"+img);
    try{
        if(urlImg == null){
            throw new RuntimeException("A imagem "+img+" não foi encontrada.");
        } else{ return ImageIO.read(urlImg); }
    } catch (IOException e) {
}

```

```

        throw new RuntimeException("Não foi possível ler a imagem:"+img+" -- "+e);
    }
}

public void paint(Graphics g) {
    Graphics2D tela = (Graphics2D) g;
    tela.setColor(Color.black);
    tela.fillRect(0, 0, getLargura(), getAltura());
    if(jogoCtrl.getFaseJogo() > jogoCtrl.getFaseFinalJogo()){
        tela.drawImage(imgFundoTela.get(Integer.toString(jogoCtrl.getFaseJogo()-1)), 0, 0, null);
    } else{
        tela.drawImage(imgFundoTela.get(Integer.toString(jogoCtrl.getFaseJogo())), 0, 0, null);
    }
    desenhaEntidade(tela);
    desenhaHUD(tela);
    desenhaMensagens(tela);
    tela.dispose();
}
private void desenhaEntidade(Graphics2D tela){
    tela.drawImage(imgJogador.get(obJogador.getImg()), obJogador.getX(),
    obJogador.getY(), null);
    Lixo ini;
    for(int j=0; j < obInimigos.size(); j++){
        ini = obInimigos.get(j);
        tela.drawImage(imgInimigos.get(ini.getImg()), ini.getX(), ini.getY(),
        null);
    }
    Tiro tiro;
    for(int i=0; i < obTiros.size(); i++){
        tiro = obTiros.get(i);
        tela.drawImage(imgTiros.get(tiro.getImg()), tiro.getX(), tiro.getY(),
        null);
    }
}
private void desenhaHUD(Graphics2D tela){
    tela.setColor(new Color(0, 0, 0, .50f));
    tela.fillRoundRect(0, 0, 255, 80, 5, 5);
    RenderingHints rh = new RenderingHints(RenderingHints.KEY_ANTIALIASING,
    RenderingHints.VALUE_ANTIALIAS_ON);
    rh.put(RenderingHints.KEY_RENDERING,RenderingHints.VALUE_RENDER_QUALITY);
    tela.setRenderingHints(rh);
    tela.setColor(Color.WHITE);
    tela.setFont(Fonte.getFonteTexto().deriveFont(Font.BOLD, 16));
    String txt = "Pontos: "+obJogador.getPontos();
    tela.drawString(txt, 15, 25);
    if(exibeMenosPontos){
        if(tempoInicioHUDJogo == 0){
            tempoInicioHUDJogo = (double)System.nanoTime()/1000000000.0; }
        double tempoHUEDevidida = (double)System.nanoTime()/1000000000.0 -
        tempoInicioHUDJogo;
        if(tempoHUEDevidida < 2){
            tela.setColor(Color.RED);
            tela.drawString("-"+Math.abs(obJogador.getPontosDiminuidos()),
            tela.getFontMetrics().stringWidth(txt)+30, 25);
        } else{
            tempoInicioHUDJogo = 0;
            exibeMenosPontos = false;
            jogoCtrl.resetarPontosDiminuidosJogador(); }
        tela.setColor(Color.WHITE);
    }
}

```

```

tela.drawString("Nível: "+jogoCtrl.getFaseJogo(), 15, 45);
tela.drawString("Capacidade do bueiro: ", 15, 65);
if(obJogador.getQtVida() <= 50){ tela.setColor(Color.RED); }
else{ tela.setColor(Color.GREEN); }
tela.drawString(+obJogador.getQtVida()+"%", 200, 65);
}

private void desenhaMensagens(Graphics2D tela){
    if(gameOver || novaFase || pause || jogoGanho){
        tela.setColor(new Color(0, 0, 0, .80f));
        tela.fillRect(0, 0, getLargura(), getAltura()); }
    if(gameOver || jogoGanho){
        tela.setColor(Color.WHITE);
        tela.setFont(Fonte.getFonteTexto().deriveFont(Font.BOLD, 30));
        if(gameOver){
            tela.drawString("Você perdeu ;(", getLargura()/2-120, getAltura()/2-30-40);
        } else if(jogoGanho){
            tela.drawString("Você ganhou o jogo! o/", getLargura()/2-160,
getAltura()/2-30-40);
        }
        tela.drawString("Sua pontuação: "+obJogador.getPontos(),
getLargura()/2-160, getAltura()/2-16);
        tela.setColor(new Color(255, 255, 255, 255));
        tela.fillRoundRect(getLargura()/2-205, getAltura()/2+8+16, 200, 50, 5,
5);
        tela.setColor(Color.BLACK);
        tela.setFont(Fonte.getFonteTexto().deriveFont(Font.BOLD, 30));
        String txt = "Novo jogo";
        tela.drawString(txt, (getLargura()/2-207)+(200/2)-
(tela.getFontMetrics().stringWidth(txt)/2), getAltura()/2+60);
        frmRtgNovoJogo = new Rectangle(getLargura()/2-205, getAltura()/2+8+16,
200, 50);
        tela.setColor(new Color(255, 255, 255, 255));
        tela.fillRoundRect(getLargura()/2+05, getAltura()/2+8+16, 230, 50, 5,
5);
    }
    if(!pontosSalvos){
        tela.setColor(Color.BLACK);
        frmRtgSalvarPontos = new Rectangle(getLargura()/2+05,
getAltura()/2+8+16, 230, 50);
    } else{
        tela.setColor(Color.GRAY);
        frmRtgSalvarPontos = new Rectangle(-10, -10, 0, 0); }
    tela.setFont(Fonte.getFonteTexto().deriveFont(Font.BOLD, 30));
    txt = "Salvar pontos";
    tela.drawString(txt, (getLargura()/2+05)+(230/2)-
(tela.getFontMetrics().stringWidth(txt)/2), getAltura()/2+60);
    } else if(novaFase){
        if(tempoInicioMsgJogo == 0){
            tempoInicioMsgJogo = (double)System.nanoTime()/1000000000.0; }
        double tempoMsgExibida = (double)System.nanoTime()/1000000000.0 -
tempoInicioMsgJogo;
        if(tempoMsgExibida < 2){
            tela.setColor(Color.WHITE);
            tela.setFont(Fonte.getFonteTexto().deriveFont(Font.BOLD, 30));
            String txt = "Próximo nível!";
            tela.drawString(txt, getLargura()/2-
(tela.getFontMetrics().stringWidth(txt))/2, getAltura()/2-21);
        } else{
            tempoInicioMsgJogo = 0;
        }
    }
}

```

```

        novaFase = false;
        pause   = false; }
    } else if(pause){
        tela.setColor(Color.WHITE);
        tela.setFont(Fonte.getFontTexto().deriveFont(Font.BOLD, 30));
        String txt = "pausado";
        tela.drawString(txt, getLargura()/2-
(tela.getFontMetrics().stringWidth(txt))/2, getAltura()/2-21);
    }
    public void salvarPontuacao(){
        SalvaPontuacao salvaPontos = new SalvaPontuacao();
        if(salvaPontos.verificaSeSalva(obJogador.getPontos())){
            JPanel panelSalvaPontos = salvaPontos.painel(obJogador.getPontos());
            JOptionPane.showOptionDialog(null, panelSalvaPontos, "Salvar
            pontuação", JOptionPane.NO_OPTION, JOptionPane.PLAIN_MESSAGE, null, new
String[]{}, "default");
            pontosSalvos = salvaPontos.isPontosSalvos();
        } else{ pontosSalvos = true; }
    }
    public void finalizar(){ janelaJogo.dispose(); }
    public void desativaJogo(){ ativo = false; }
    public int getLargura() { return this.getWidth(); }
    public int getAltura() { return this.getHeight(); }
    public boolean isAtivo() { return ativo; }
    public void setAtivo(boolean ativo) { this.ativo = ativo; }
    public boolean isPause() { return pause; }
    public void setPause(boolean pause) { this.pause = pause; }
    public boolean isGameOver() { return gameOver; }
    public void setGameOver(boolean gameOver) { this.gameOver = gameOver; }
    public ArrayList<Lixo> getObInimigos() { return obInimigos; }
    public ArrayList<Tiro> getObTiros() { return obTiros; }
    public Jogador getObJogador() { return obJogador; }
    public Rectangle getFrmRtgNovoJogo() { return frmRtgNovoJogo; }
    public Rectangle getFrmRtgSalvarPontos() { return frmRtgSalvarPontos; }
    public JogoCtrl getJogoCtrl() { return jogoCtrl; }
    public boolean isNovaFase() { return novaFase; }
    public void setNovaFase(boolean novaFase) { this.novaFase = novaFase; }
    public boolean isJogoGanho() { return jogoGanho; }
    public void setJogoGanho(boolean jogoGanho){ this.jogoGanho = jogoGanho; }
    public boolean isExibeMenosPontos() { return exibeMenosPontos; }
    public void setExibeMenosPontos(boolean exibeMenosPontos) {
        this.exibeMenosPontos = exibeMenosPontos; }
}

```

Fonte: Elaborado pelos autores

Programa 4 – Classe JogoCtrl

```

package control;
public class JogoCtrl {
    private Jogo obJogo;
    private InputCtrl inputCtrl;
    private int qtFilaInimigoFase      = 20;
    private int qtInimigoFila          = 1;
    private int qtInimigoFilaMin       = 1;
    private int velocidadeInimigo     = 5;
    private int espacoVertFilaIni     = 130;
    private int contePixelLimpo = espacoVertFilaIni;
    private int qtFilaInimigoAdd = 0;
}

```

```

private int contVelInimigo = 0;
private int yFilaInimigo = 0;
private int faseJogo = 1;
private int faseFinalJogo = 5;
private int qtInimigoFilaMax;
private int espacoEntreInim = 5;
private int qtFilaAdd;
private ArrayList<Integer> coordenadaInimigo = new ArrayList<Integer>();
public JogoCtrl(Jogo jogo){
    obJogo = jogo;
    geraPosicaoInimigo();
}
public void novoJogo(){
    resetaConfigFase();
    resetaConfigJogo();
    obJogo.novoJogo();
}
private void novaFaseJogo(){
    if(faseJogo <= qtInimigoFilaMax){
        qtInimigoFila = faseJogo;
        qtInimigoFilaMin = faseJogo; }
    if(faseJogo == 3){ velocidadeInimigo += 1.5; }
    else if(faseJogo == 4){ velocidadeInimigo += 1.0; }
    else if(faseJogo == 5){ velocidadeInimigo += 0.5; }
    resetaConfigFase();
    obJogo.novaFase();
    obJogo.setNovaFase(true);
    obJogo.setPause(true);
}
private void resetaConfigFase(){
    contePixelLimpo = espacoVertFilaIni;
    qtFilaInimigoAdd = 0;
    contVelInimigo = 0;
    yFilaInimigo = 0;
}
private void resetaConfigJogo(){
    qtInimigoFila = 1;
    qtInimigoFilaMin = 1;
    velocidadeInimigo= 5;
    faseJogo = 1;
}
public void setInputCtrl(InputCtrl inputCtrl){
    this.inputCtrl = inputCtrl;
}
public void atualizar(){
    atualizaTiros();
    if(contVelInimigo == velocidadeInimigo){
        atualizaInimigos();
        adicionaInimigos();
        contVelInimigo = 0;
    }
    contVelInimigo++;
    verificaColisoes();
    verificaTeclas();
    verificaGameOver();
    verificaNovaFase();
    verificaJogoGanho();
    Thread.yield();
}

```

```

public void verificaTeclas(){
    HashMap<Integer, Boolean> tecla = inputCtrl.getTeclaPressionada();
    if(!objJogo.isPause()){
        if(tecla.get(KeyEvent.VK_LEFT) != null){ atualizaJogador("esq"); }
        if(tecla.get(KeyEvent.VK_RIGHT) != null){ atualizaJogador("dir"); }
        if(tecla.get(KeyEvent.VK_SPACE) != null){ atiraMissel(); }
        tecla.remove(KeyEvent.VK_SPACE); }
        if(tecla.get(KeyEvent.VK_A) != null){ atualizaTipoTiro("A"); }
        if(tecla.get(KeyEvent.VK_S) != null){ atualizaTipoTiro("S"); }
        if(tecla.get(KeyEvent.VK_D) != null){ atualizaTipoTiro("D"); }
        if(tecla.get(KeyEvent.VK_F) != null){ atualizaTipoTiro("F"); }
        if(tecla.get(KeyEvent.VK_G) != null){ atualizaTipoTiro("G"); }
    }
}

public void verificaTeclasCtrlJanela(){
    HashMap<Integer, Boolean> tecla = inputCtrl.getTeclaPressionada();
    if(tecla.get(KeyEvent.VK_P) != null){
        pausaJogo();
        tecla.clear(); }
    if(tecla.get(KeyEvent.VK_ESCAPE) != null){
        objJogo.desativaJogo();
        objJogo.finalizar(); }
    if(objJogo.isGameOver() || objJogo.isJogoGanho()){
        if(tecla.get(KeyEvent.VK_ENTER) != null){ novoJogo(); }
    }
}

private int qtMaxInimigoFila(Lixo inimigo){
    qtInimigoFilaMax = 5;
    return objJogo.getLargura() - (inimigo.getLargura() + espacoEntreInim) * qtInimigoFilaMax;
}

private void geraPosicaoInimigo(){
    Lixo inimigo = new Lixo();
    int restoPixel = qtMaxInimigoFila(inimigo);
    int espacoExtra = restoPixel / qtInimigoFilaMax;
    int espacoExtraResto = espacoExtra % qtInimigoFilaMax;
    int j = 0;
    int posicao, espacoEsq;
    for(int i=0; i<qtInimigoFilaMax; i++){
        if(espacoExtraResto > 0){
            espacoExtraResto--;
            j = 1; } else{ j = 0; }
        posicao = i * (inimigo.getLargura() + espacoEntreInim) + (espacoExtra * i) + j;
        espacoEsq = espacoExtra / 2;
        coordenadaInimigo.add(posicao + espacoEsq); }
    }

public void pausaJogo(){
    if(objJogo.isPause()){ objJogo.setPause(false); }
    else{ objJogo.setPause(true); }
}

private void verificaGameOver(){
    if(objJogo.getObjJogador().getQtVida() <= 0){ objJogo.setGameOver(true); }
}

private void verificaNovaFase(){
    if(qtFilaInimigoAdd == qtFilaInimigoFase){
        if(objJogo.getObjInimigos().size() == 0){
            faseJogo++;
            novaFaseJogo(); }
    }
}

private void verificaJogoGanho(){
    if(faseJogo == faseFinalJogo + 1 && objJogo.getObjInimigos().size() == 0){
}

```

```

        obJogo.setJogoGanho(true);
        obJogo.setPause(true);
    }
    private void atualizaJogador(String direcao){
        Jogador j = obJogo.getObJogador();
        if(j.getX() > 0 && direcao.equals("esq")){ j.setX(j.getX()-1); }
        if(j.getX() < obJogo.getLargura()-j.getLargura() &&
direcao.equals("dir")){ j.setX(j.getX()+1); }
    }
    private void atualizaTipoTiro(String sentido){
        ArrayList<String> tiposTiro = TipoMaterial.getTipos();
        switch(sentido){
            case "A": obJogo.getObJogador().setTipoTiro(tiposTiro.get(0)); break;
            case "S": obJogo.getObJogador().setTipoTiro(tiposTiro.get(1)); break;
            case "D": obJogo.getObJogador().setTipoTiro(tiposTiro.get(2)); break;
            case "F": obJogo.getObJogador().setTipoTiro(tiposTiro.get(3)); break;
            case "G": obJogo.getObJogador().setTipoTiro(tiposTiro.get(4)); break;
        }
    }
    private void atualizaTiros() {
        ArrayList<Tiro> obT = obJogo.getObTiros();
        for(int i=0; i < obT.size(); i++){
            Tiro t = obT.get(i);
            if(t.getY() < 0){ obT.remove(i); } else{ t.setY(t.getY()-1); }
        }
    }
    private void atiraMissel(){
        if(verificaSeAtira()){
            obJogo.getObTiros().add(new Tiro(obJogo.getObJogador().getX()
+obJogo.getObJogador().getLargura()-Tiro.imgLargura, obJogo.getAltura()-
obJogo.getObJogador().getAltura(), obJogo.getObJogador().getTipoTiro()));
        }
    }
    private boolean verificaSeAtira(){
        if(obJogo.getObTiros().size() == 0){ return true; } else{
            Tiro ultimoTiro = (Tiro)
obJogo.getObTiros().get(obJogo.getObTiros().size()-1);
            if(ultimoTiro.getY() < obJogo.getAltura()-ultimoTiro.getAltura()-
obJogo.getObJogador().getAltura()){ return true; } else{ return false; }
        }
    }
    private void adicionaInimigos(){
        if(qtFilaInimigoAdd < qtFilaInimigoFase){
            if(contePixelLimpo == espacoVertFilaIni){
                geraFilaInimigos();
                qtFilaInimigoAdd++;
                contePixelLimpo = 0; }
            contePixelLimpo++;
        }
    }
    private void geraFilaInimigos(){
        Random rand;
        ArrayList coordJaUsada = new ArrayList();
        rand = new Random();
        int qtInimFila = rand.nextInt((qtInimigoFila - qtInimigoFilaMin) + 1) +
qtInimigoFilaMin;
        int espacoExtra;
        int j = 0;
        if(qtFilaAdd == 0){
            espacoExtra = 0;
            qtFilaAdd++;
        } else{
            if(qtInimigoFila == qtInimigoFilaMax){ j = 1; }
            espacoExtra= coordenadaInimigo.get(1)/2;
        }
    }
}

```

```

        qtFilaAdd = 0; }
    for(int i = j; i<qtInimFila; i++){
        rand = new Random();
        int numAlea = rand.nextInt((TipoMaterial.getTipos().size()-1) + 1) + 0;
        obJogo.getObInimigos().add(new
Lixo(TipoMaterial.getTipos().get(numAlea)));
        rand = new Random();
        int xInimigo;
        do{
            if(qtFilaAdd == 0){
                xInimigo = rand.nextInt((coordenadaInimigo.size()-2)-0) + 1) + 0;
            } else{
                xInimigo = rand.nextInt((coordenadaInimigo.size()-1)-0) + 1) +0; }
        }
        while(coordJaUsada.contains(coordenadaInimigo.get(xInimigo)));
        obJogo.getObInimigos().get(obJogo.getObInimigos().size()-
1).setX(coordenadaInimigo.get(xInimigo)+espacoExtra);
        obJogo.getObInimigos().get(obJogo.getObInimigos().size()-
1).setY(yFilaInimigo);
        coordJaUsada.add(coordenadaInimigo.get(xInimigo));
    }
}

private void atualizaInimigos(){
    for(int j=0; j < obJogo.getObInimigos().size(); j++){
        Lixo l = obJogo.getObInimigos().get(j);
        if(obJogo.getAltura() > l.getY()){ l.setY(l.getY()+1); } else{
            obJogo.getObJogador().setQtVida(-l.getQtVidaRetirada());
            obJogo.getObJogador().setPontos(-l.getPontos());
            obJogo.setExibeMenosPontos(true);
            obJogo.getObInimigos().remove(j);
        }
    }
}

private void verificaColisoes(){
    Rectangle formaTiro;
    Rectangle formaInimigo;
    Tiro tiro;
    Lixo ini;
    for(int i=0; i < obJogo.getObTiros().size(); i++){
        tiro      = obJogo.getObTiros().get(i);
        formaTiro = tiro.getForma();
        for(int j=0; j < obJogo.getObInimigos().size(); j++){
            ini = obJogo.getObInimigos().get(j);
            formaInimigo = ini.getForma();
            if(formaTiro.intersects(formaInimigo)){
                if(tiro.getTipo().equals(ini.getTipo())){
                    obJogo.getObJogador().setPontos(ini.getPontos());
                    obJogo.getObInimigos().remove(j);
                } else{
                    obJogo.getObJogador().setPontos(-ini.getPontos());
                    obJogo.setExibeMenosPontos(true);
                    obJogo.getObTiros().remove(i); break;
                }
            }
        }
    }
}

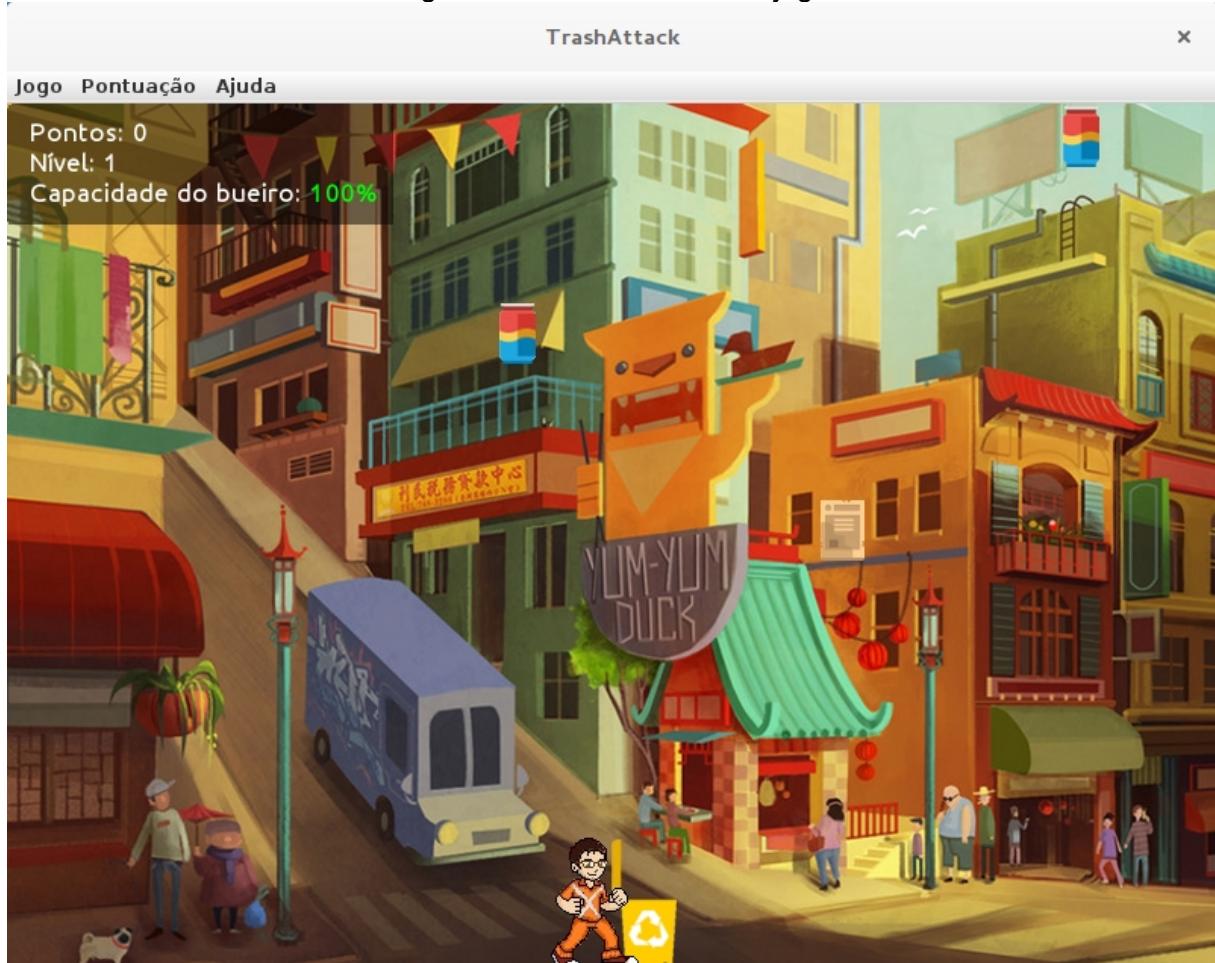
public int getFaseJogo() { return faseJogo; }
public int getFaseFinalJogo() { return faseFinalJogo; }
public void setFaseFinalJogo(int faseFinalJogo) {
    this.faseFinalJogo = faseFinalJogo; }
public void resetarPontosDiminuidosJogador(){
    obJogo.getObJogador().setPontosDiminuidos(0); }
}

```

Fonte: Elaborado pelos autores

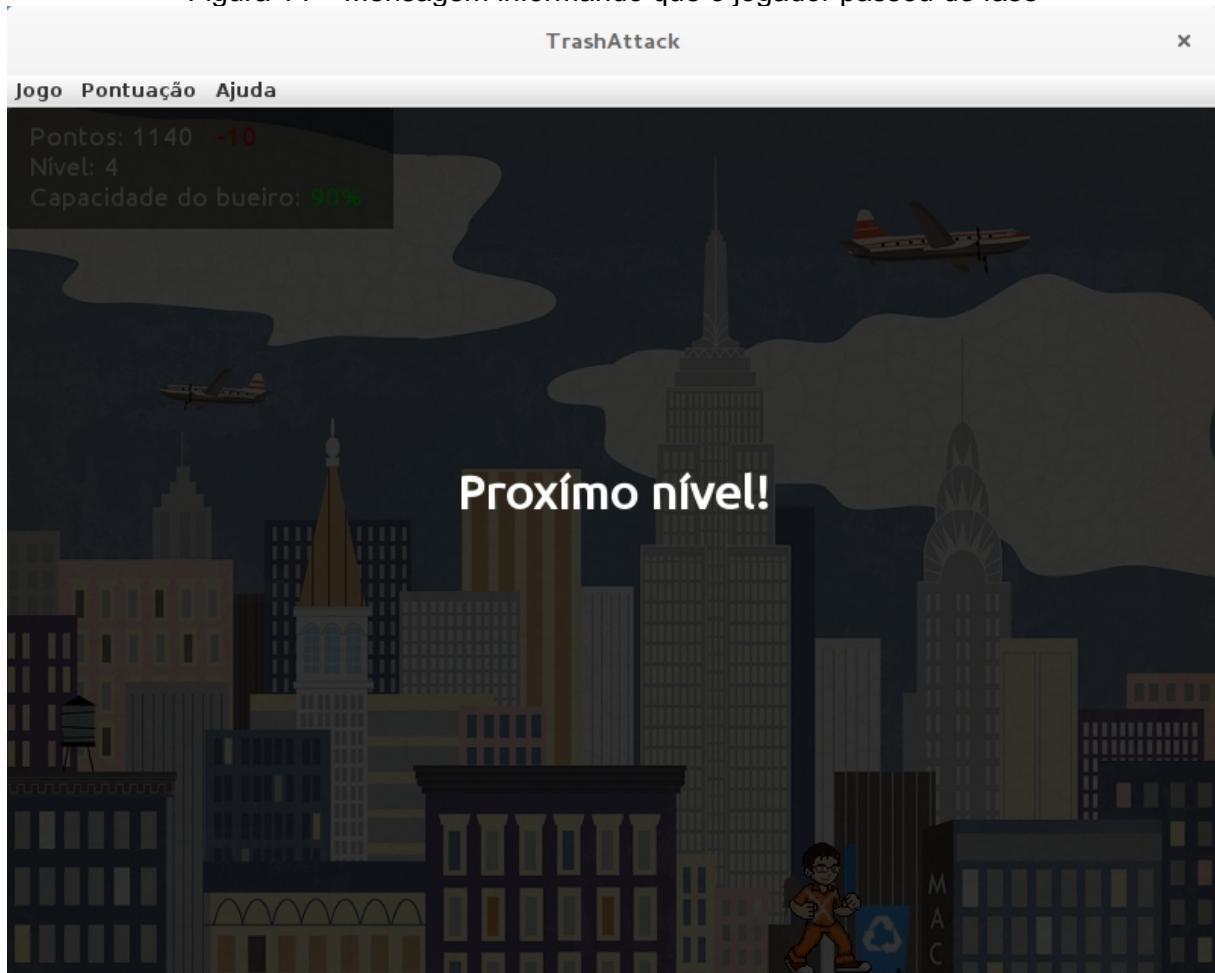
7 APRESENTAÇÃO DO JOGO EM FUNCIONAMENTO EM UM COMPUTADOR

Figura 13 – Primeira fase do jogo



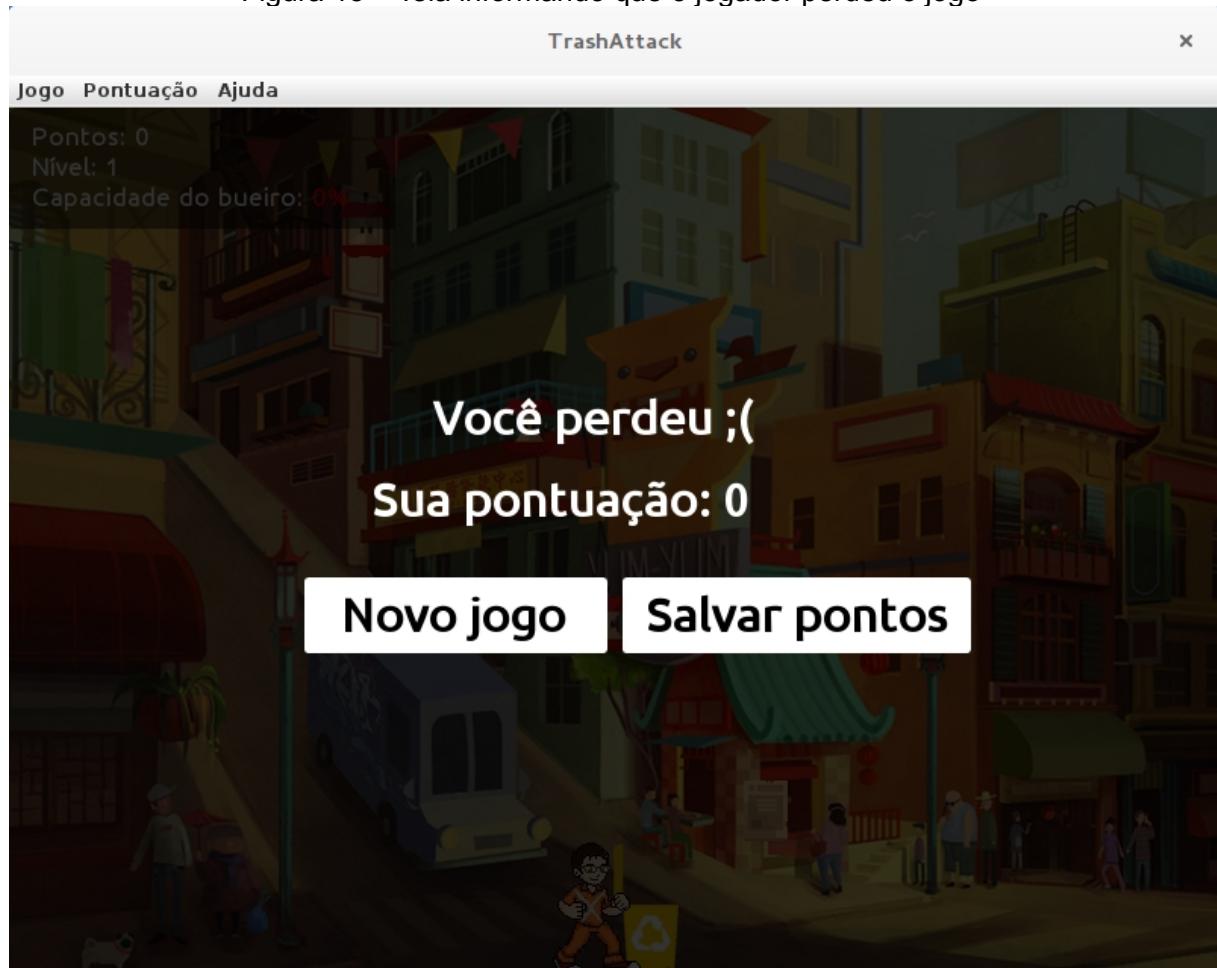
Fonte: Elaborada pelos autores

Figura 14 – Mensagem informando que o jogador passou de fase



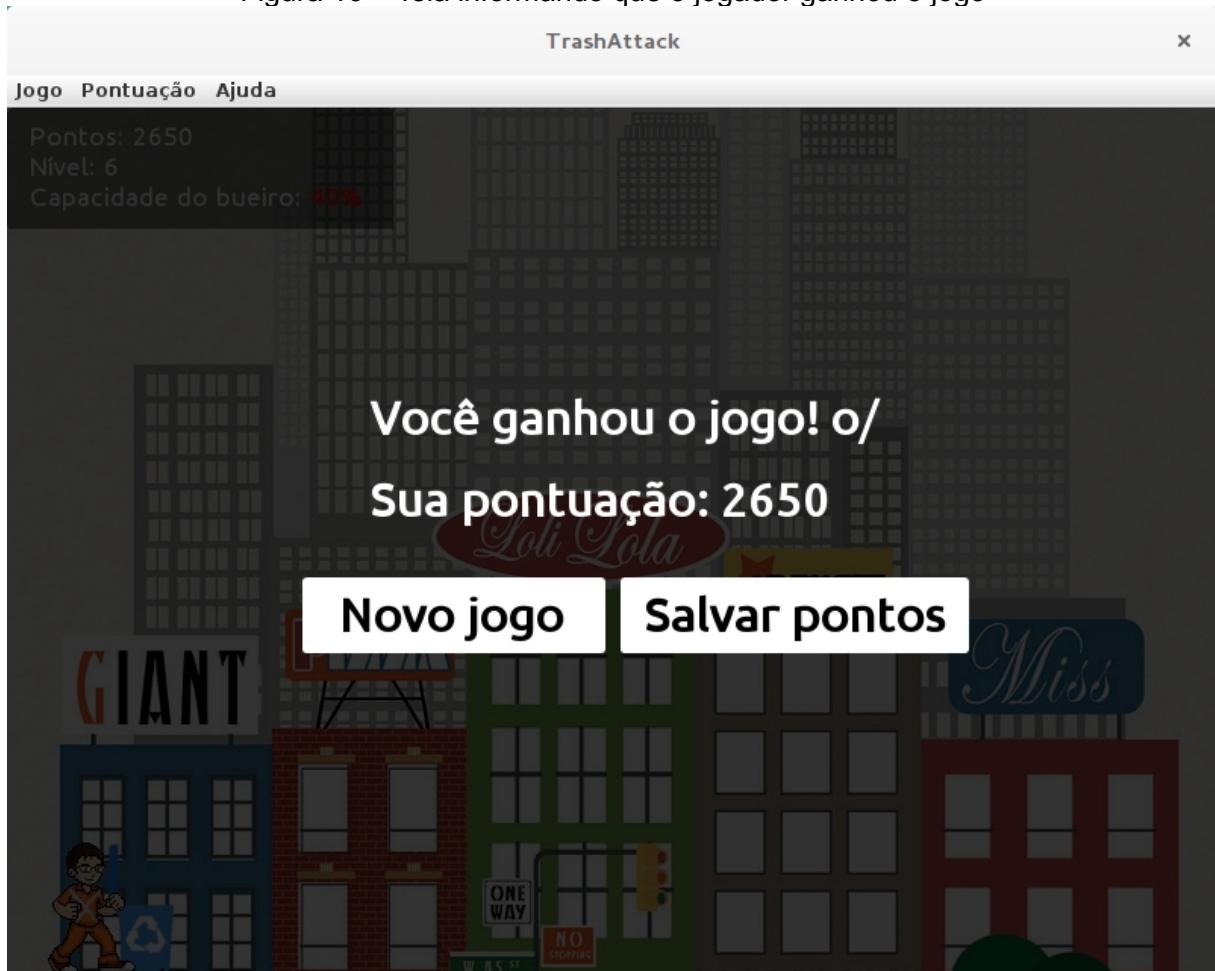
Fonte: Elaborada pelos autores

Figura 15 – Tela informando que o jogador perdeu o jogo



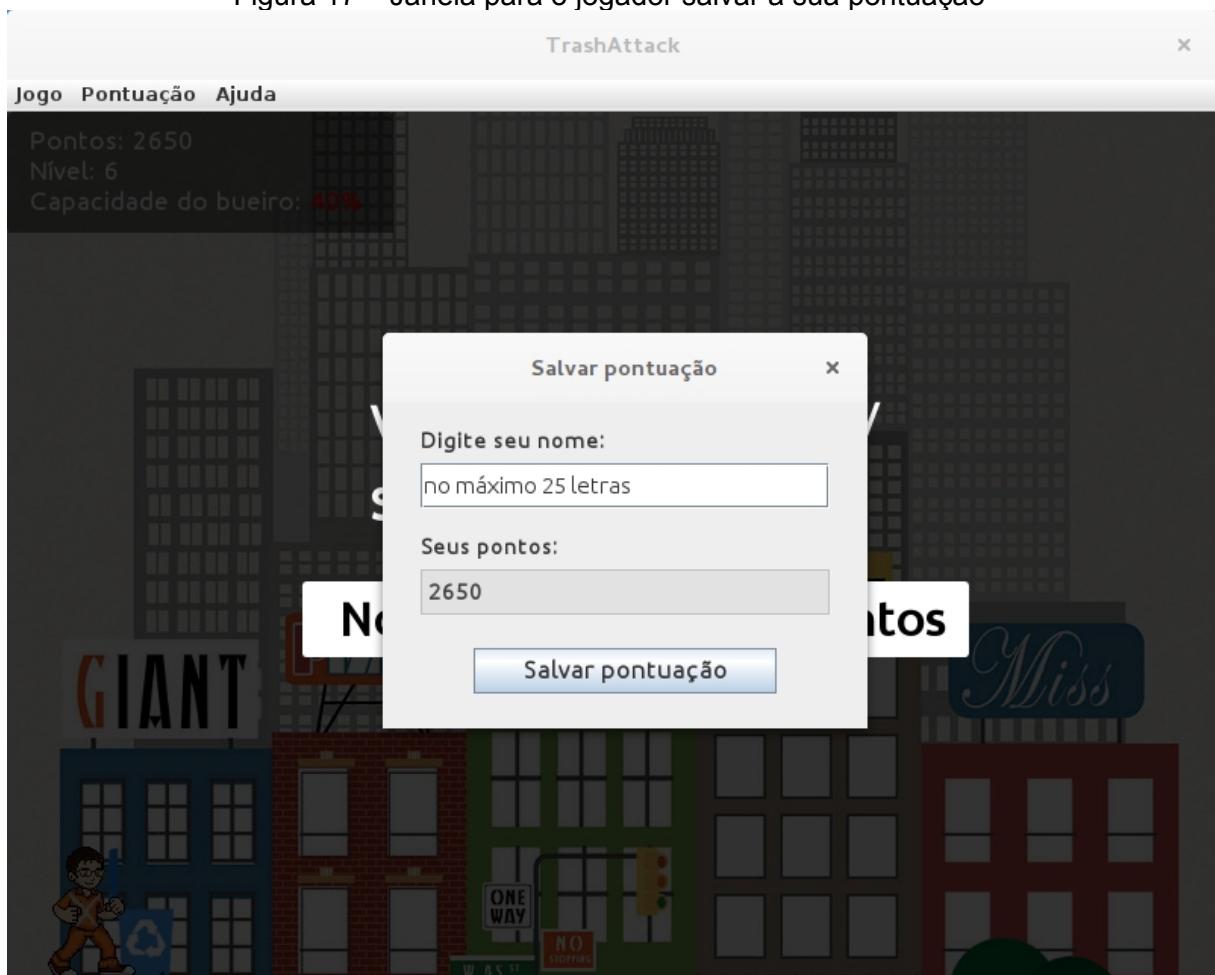
Fonte: Elaborada pelos autores

Figura 16 – Tela informando que o jogador ganhou o jogo



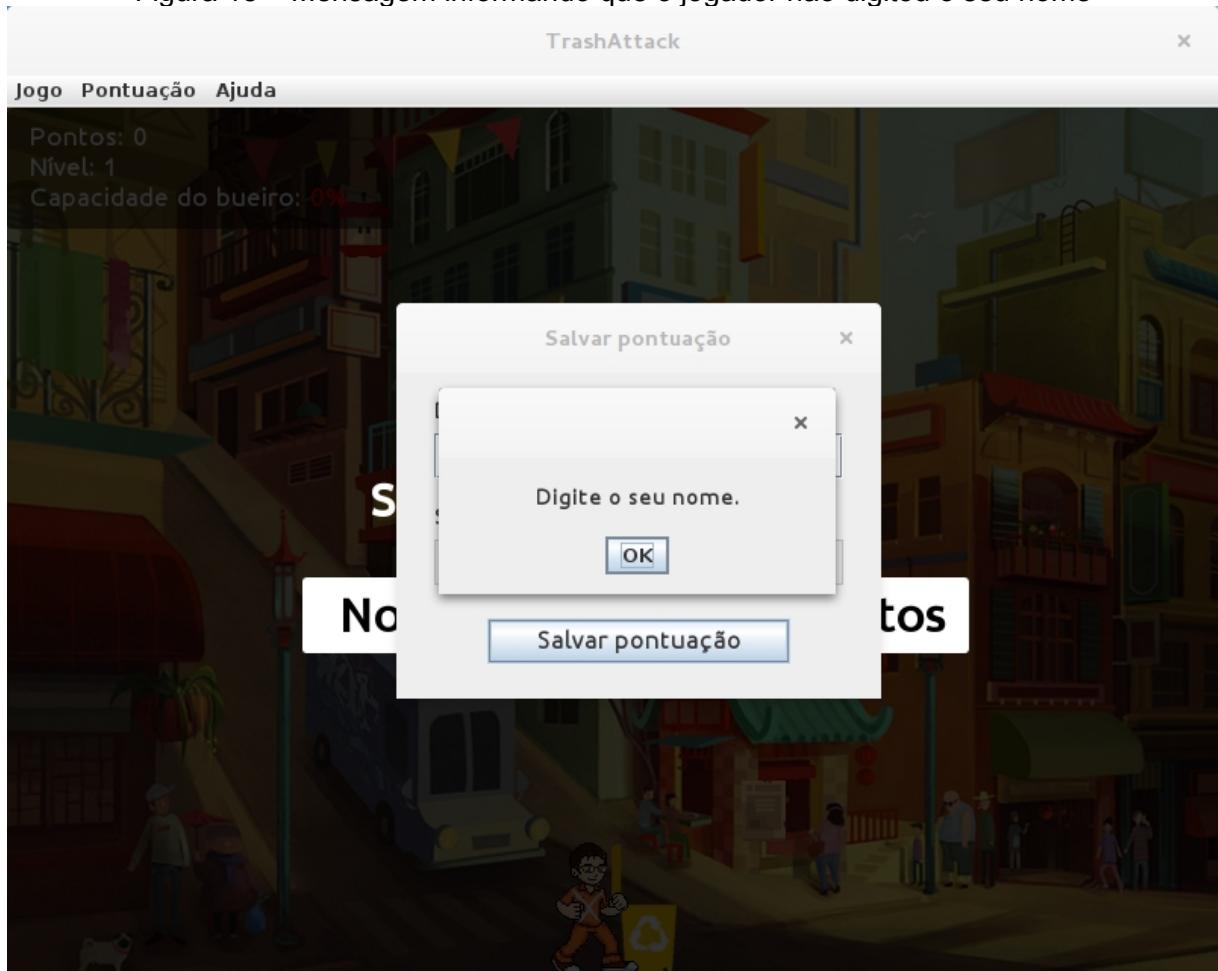
Fonte: Elaborada pelos autores

Figura 17 – Janela para o jogador salvar a sua pontuação



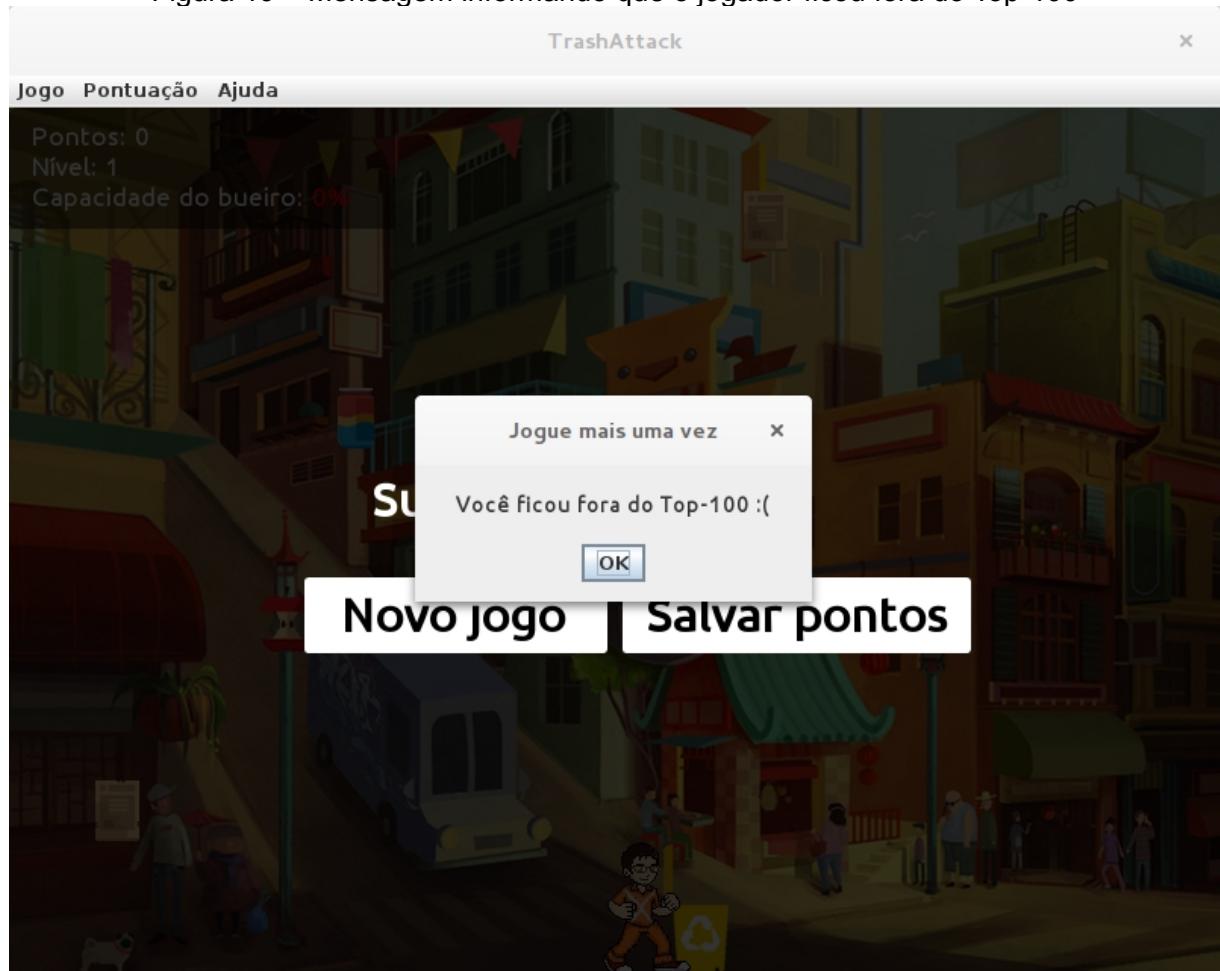
Fonte: Elaborada pelos autores

Figura 18 – Mensagem informando que o jogador não digitou o seu nome



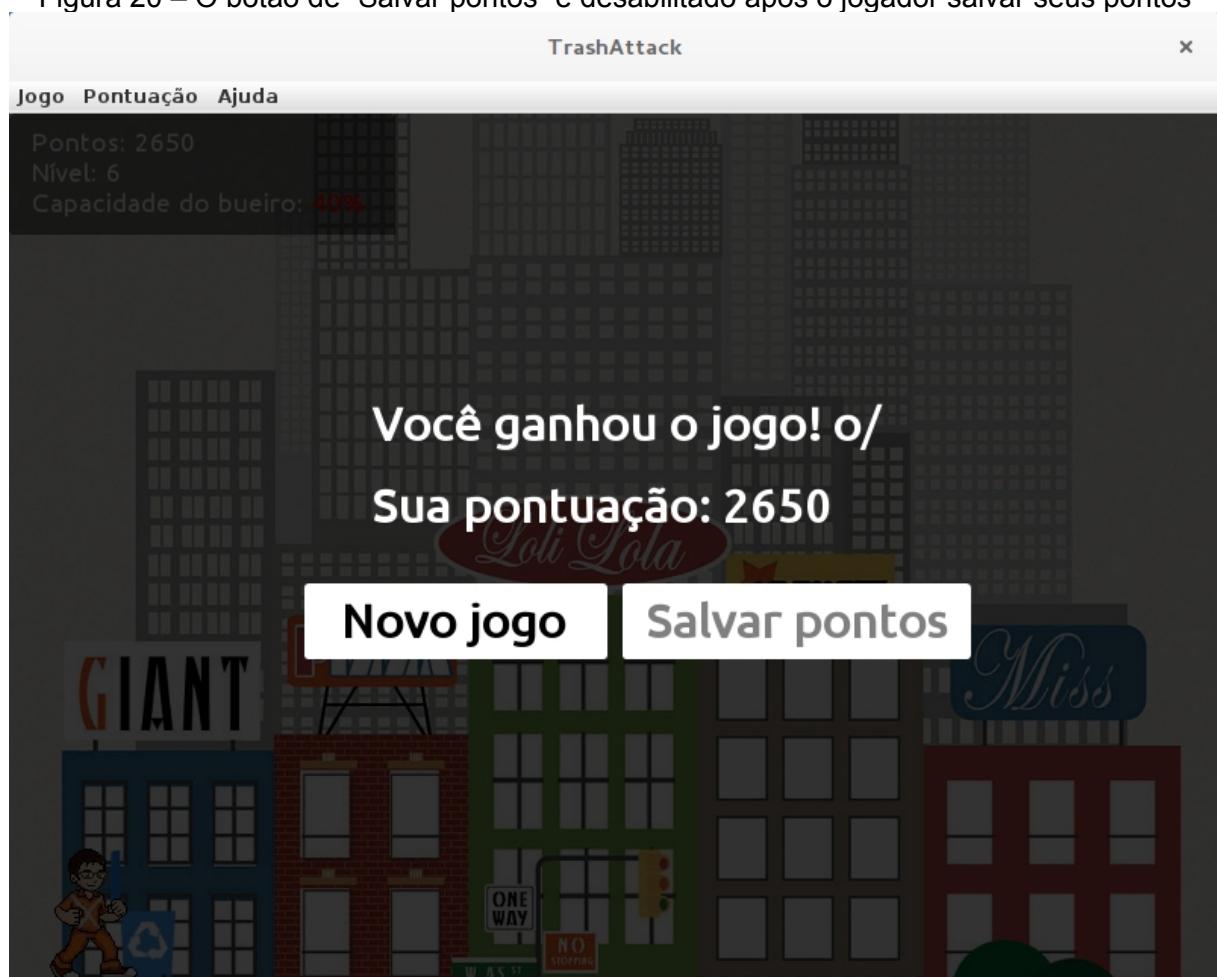
Fonte: Elaborada pelos autores

Figura 19 – Mensagem informando que o jogador ficou fora do Top-100



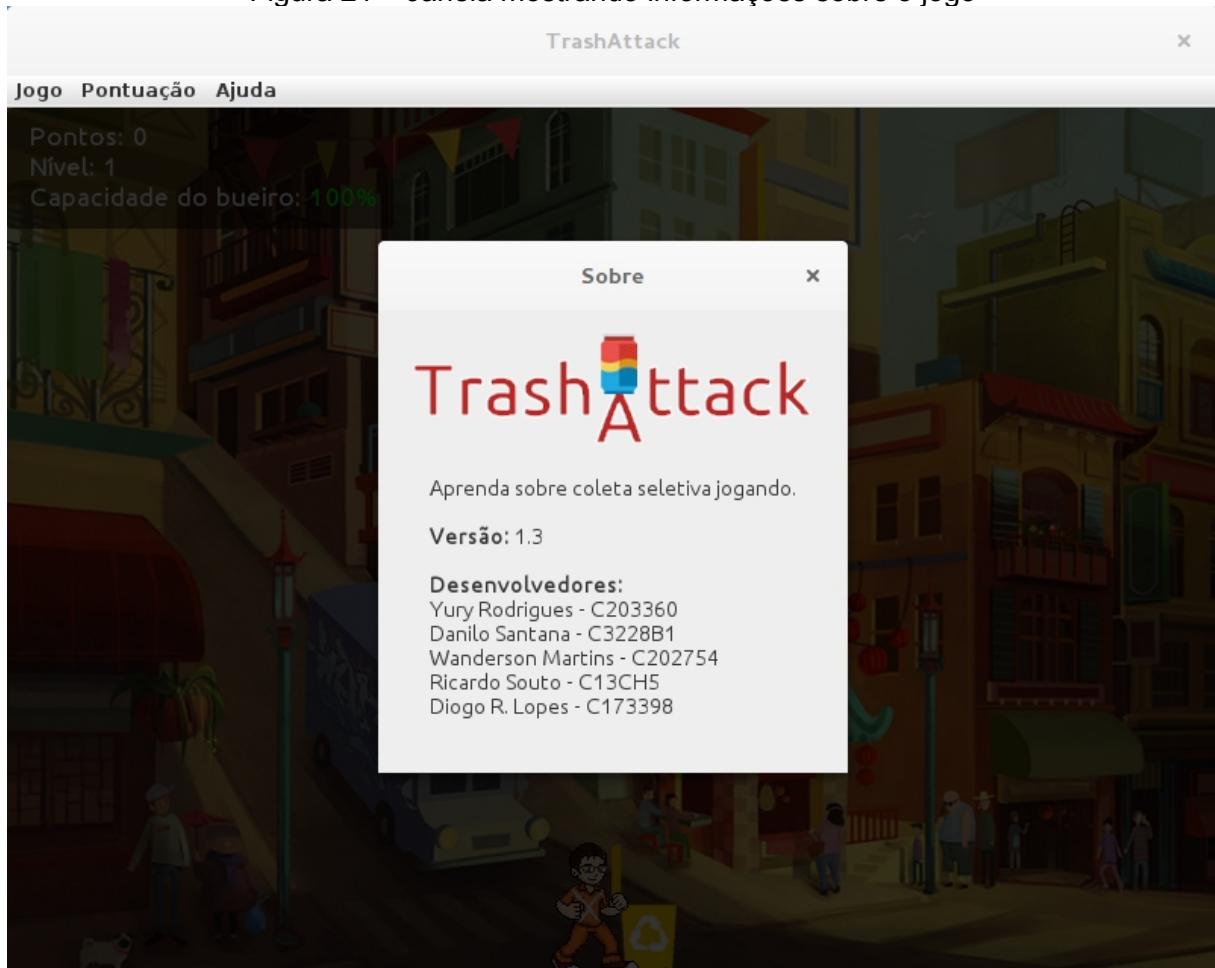
Fonte: Elaborada pelos autores

Figura 20 – O botão de “Salvar pontos” é desabilitado após o jogador salvar seus pontos



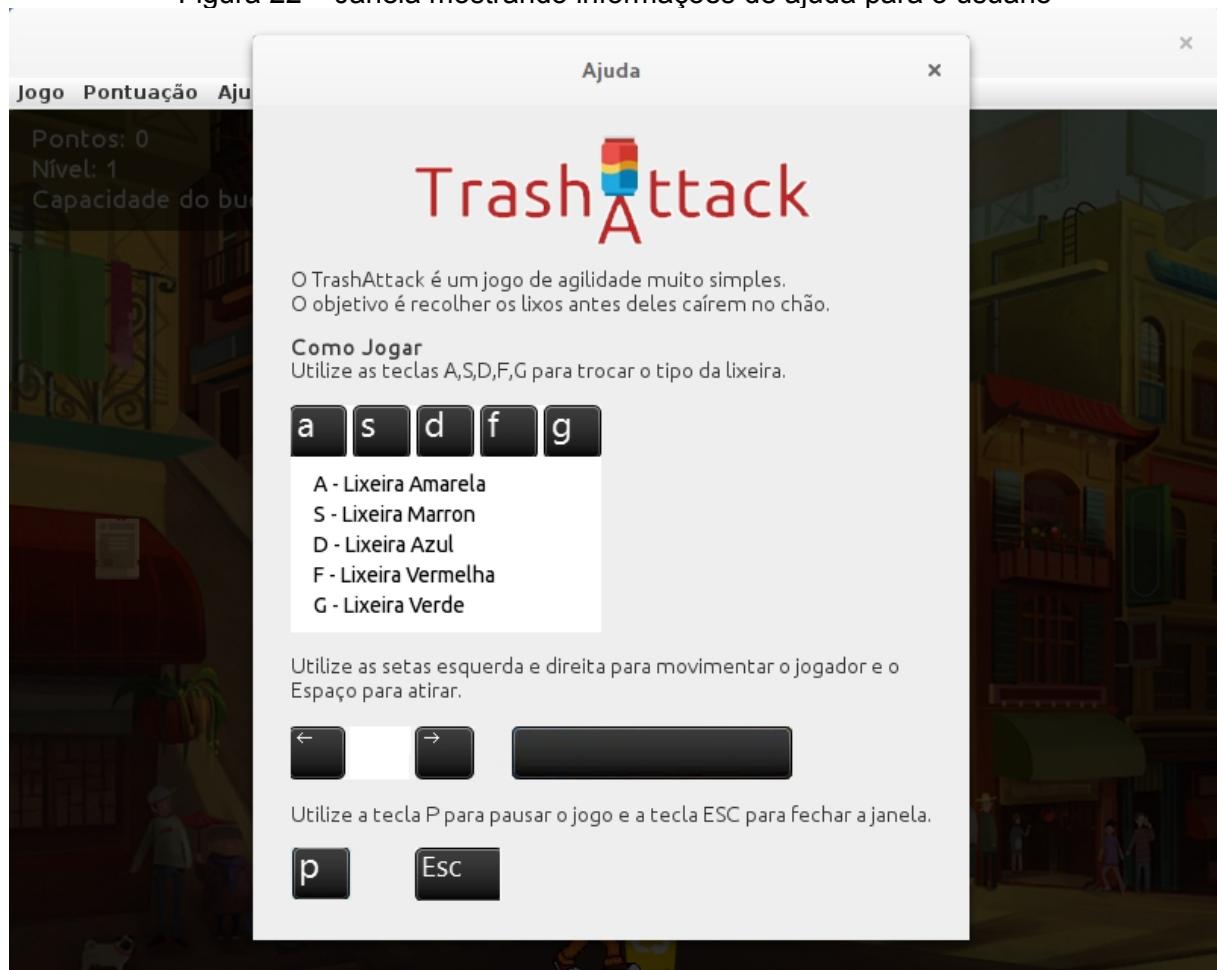
Fonte: Elaborada pelos autores

Figura 21 – Janela mostrando informações sobre o jogo



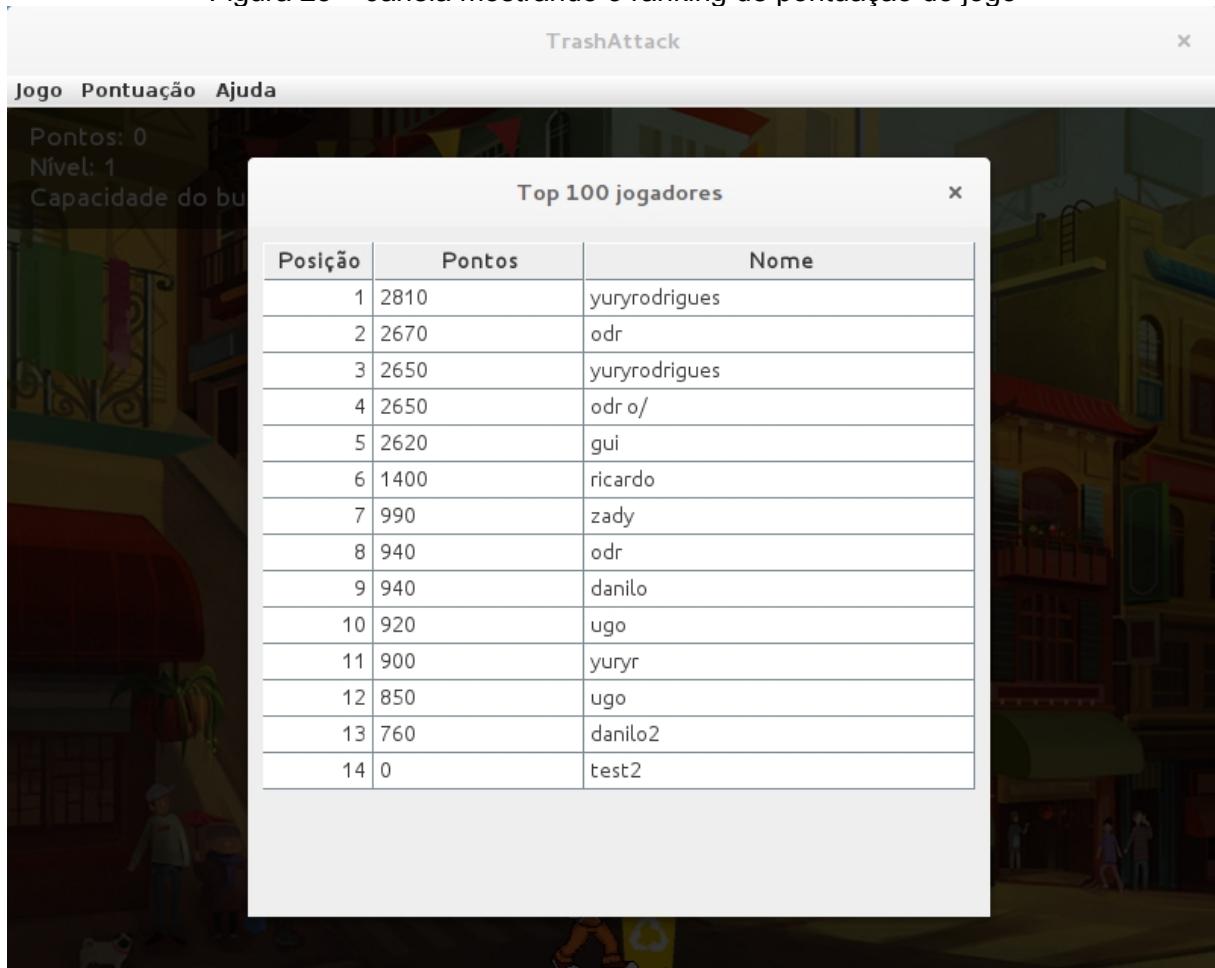
Fonte: Elaborada pelos autores

Figura 22 – Janela mostrando informações de ajuda para o usuário



Fonte: Elaborada pelos autores

Figura 23 – Janela mostrando o ranking de pontuação do jogo

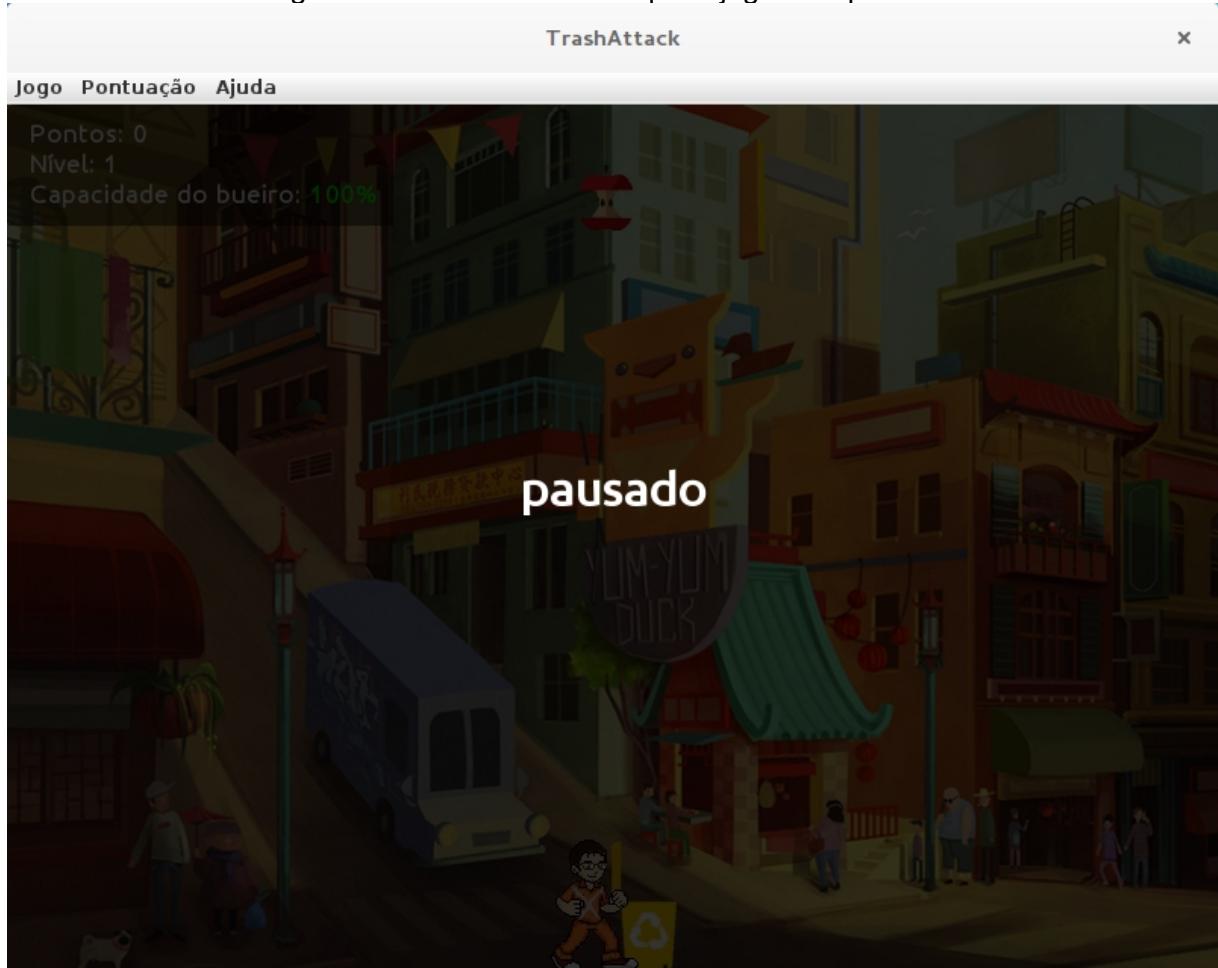


The image shows a screenshot of the game 'TrashAttack'. A modal window titled 'Top 100 jogadores' (Top 100 players) is displayed in the center. The window has three columns: 'Posição' (Position), 'Pontos' (Points), and 'Nome' (Name). The data is as follows:

Posição	Pontos	Nome
1	2810	yuryrodrigues
2	2670	odr
3	2650	yuryrodrigues
4	2650	odr o/
5	2620	gui
6	1400	ricardo
7	990	zady
8	940	odr
9	940	danilo
10	920	ugo
11	900	yuryr
12	850	ugo
13	760	danilo2
14	0	test2

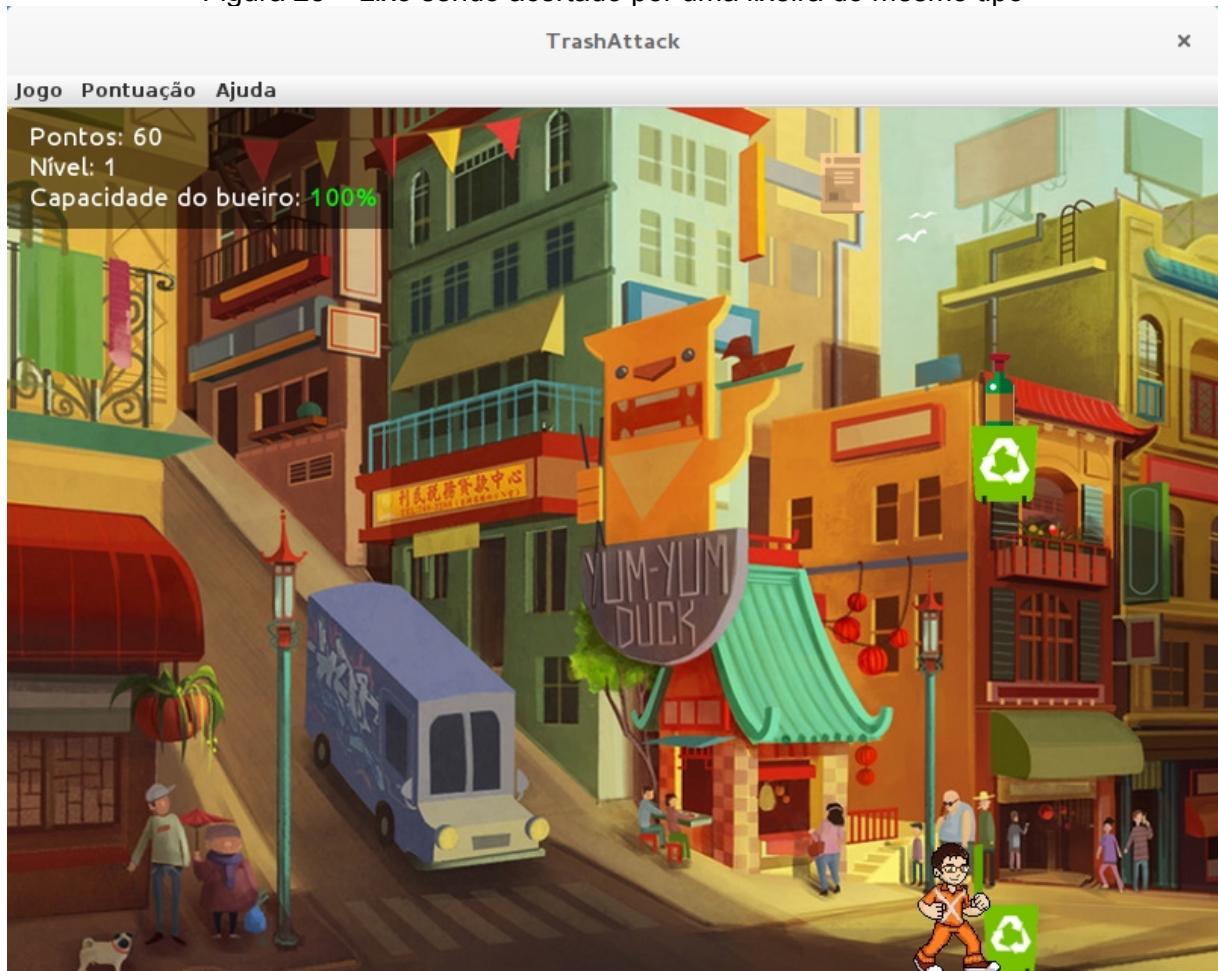
Fonte: Elaborada pelos autores

Figura 24 – Tela informando que o jogo está pausado



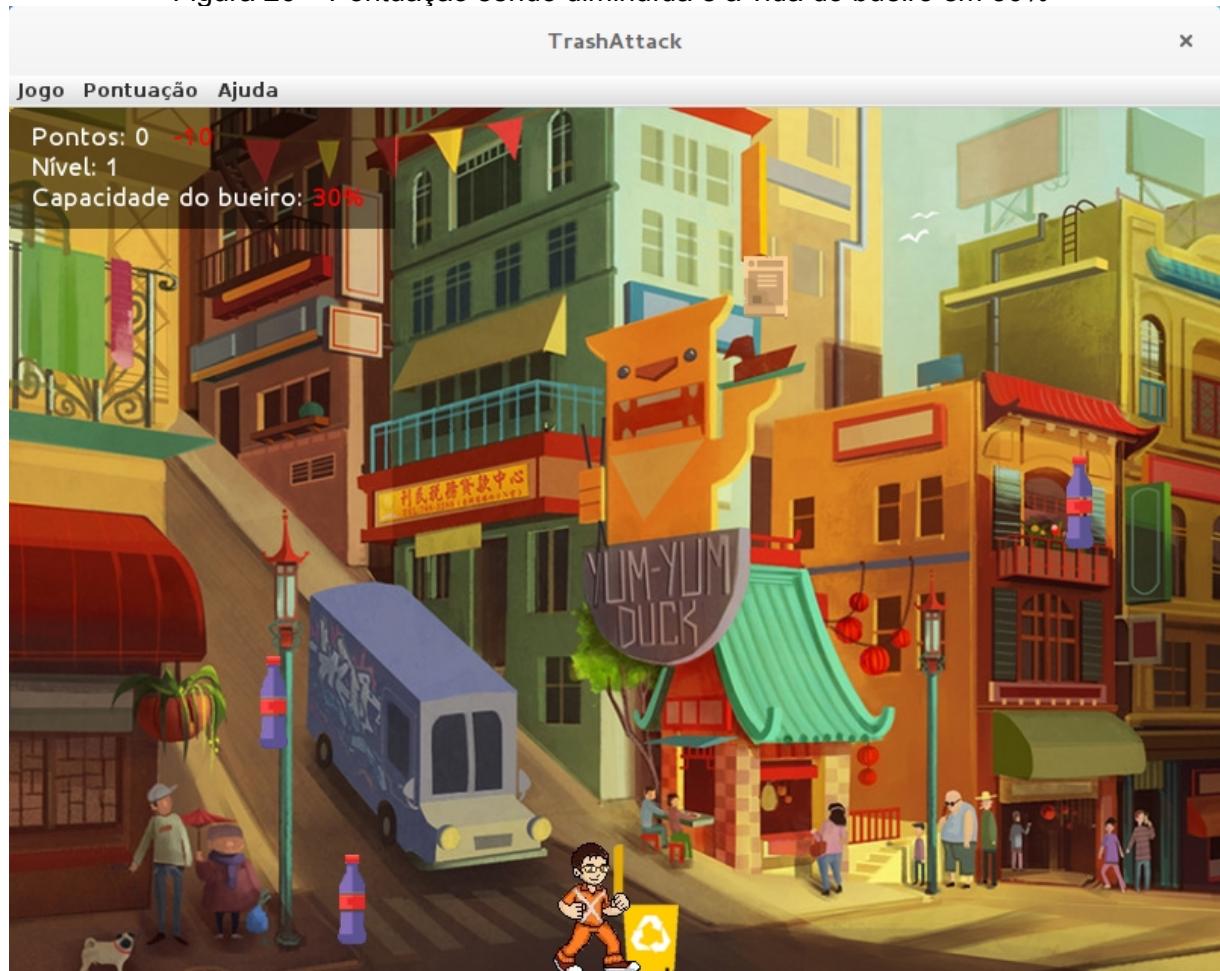
Fonte: Elaborada pelos autores

Figura 25 – Lixo sendo acertado por uma lixeira do mesmo tipo



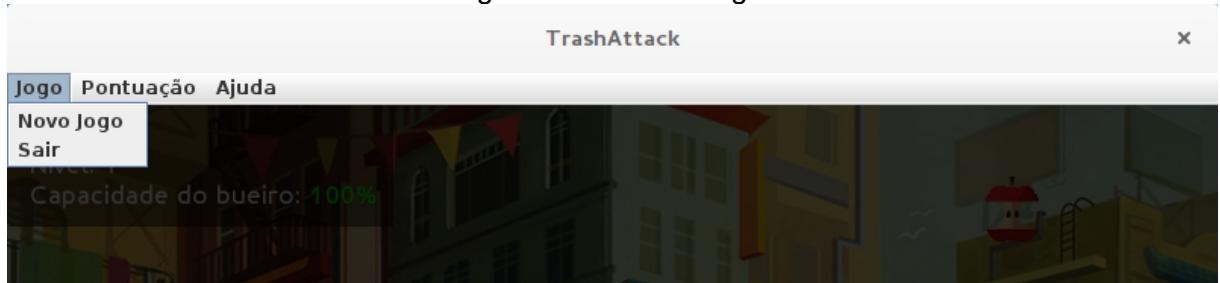
Fonte: Elaborada pelos autores

Figura 26 – Pontuação sendo diminuída e a vida do bueiro em 30%



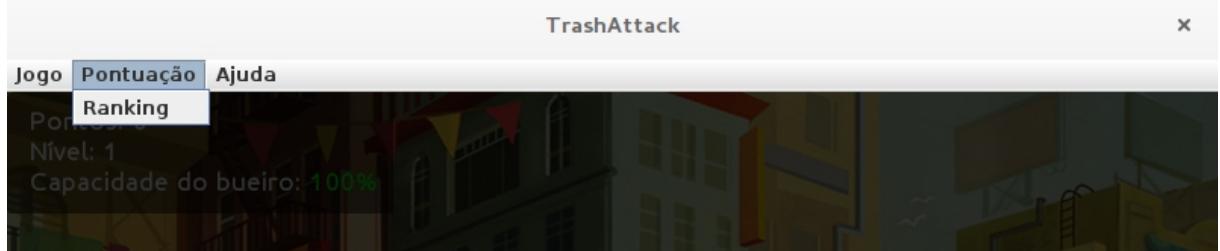
Fonte: Elaborada pelos autores

Figura 27 – Menu “Jogo”



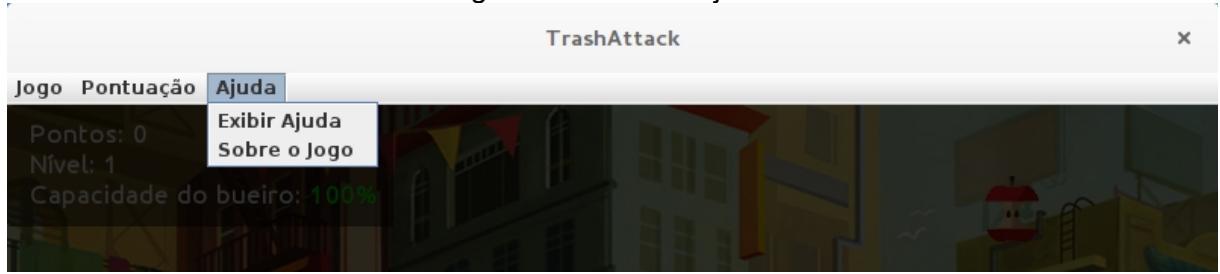
Fonte: Elaborada pelos autores

Figura 28 – Menu “Pontuação”



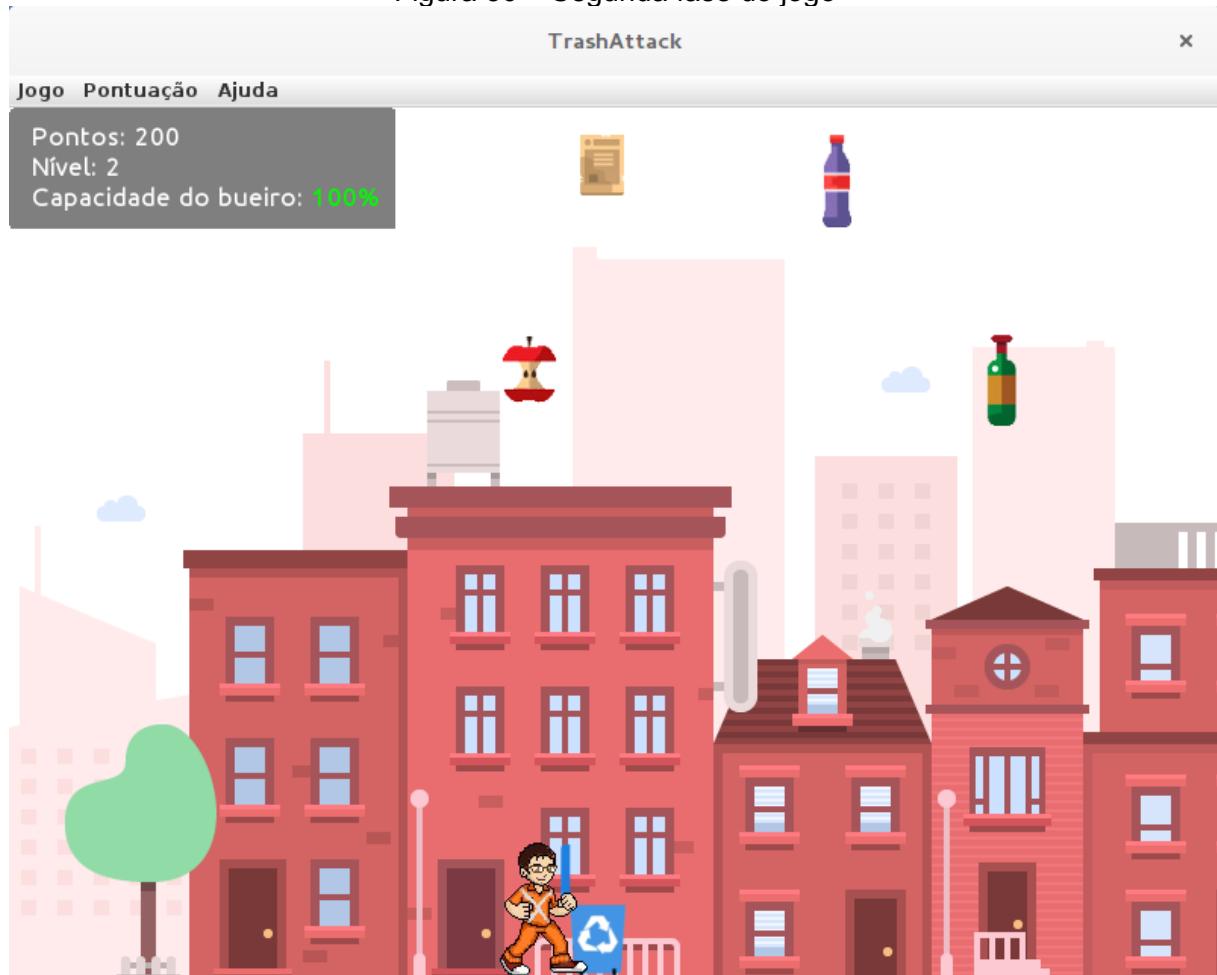
Fonte: Elaborada pelos autores

Figura 29 – Menu “Ajuda”



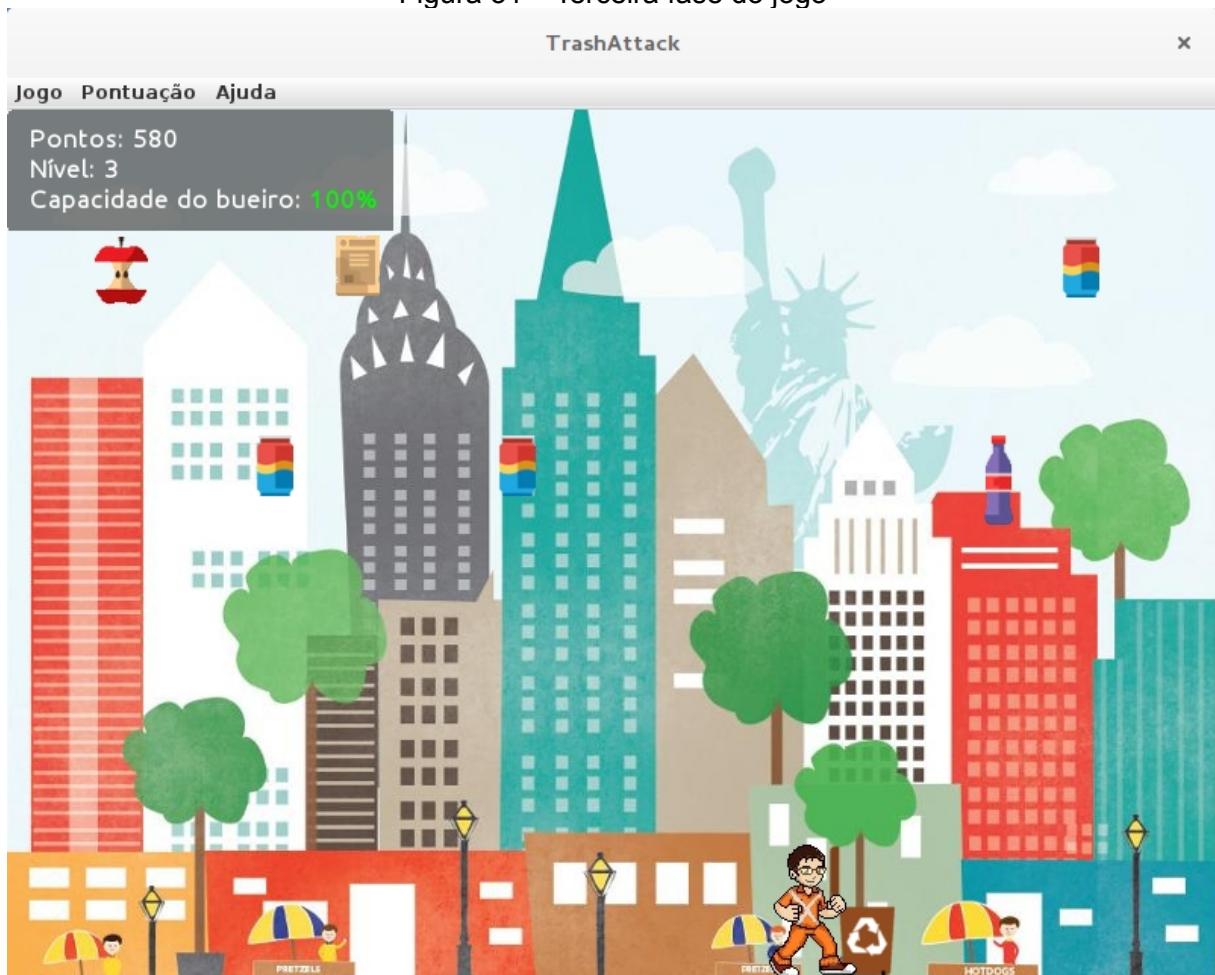
Fonte: Elaborada pelos autores

Figura 30 – Segunda fase do jogo



Fonte: Elaborada pelos autores

Figura 31 – Terceira fase do jogo



Fonte: Elaborada pelos autores

Figura 32 – Quarta fase do jogo

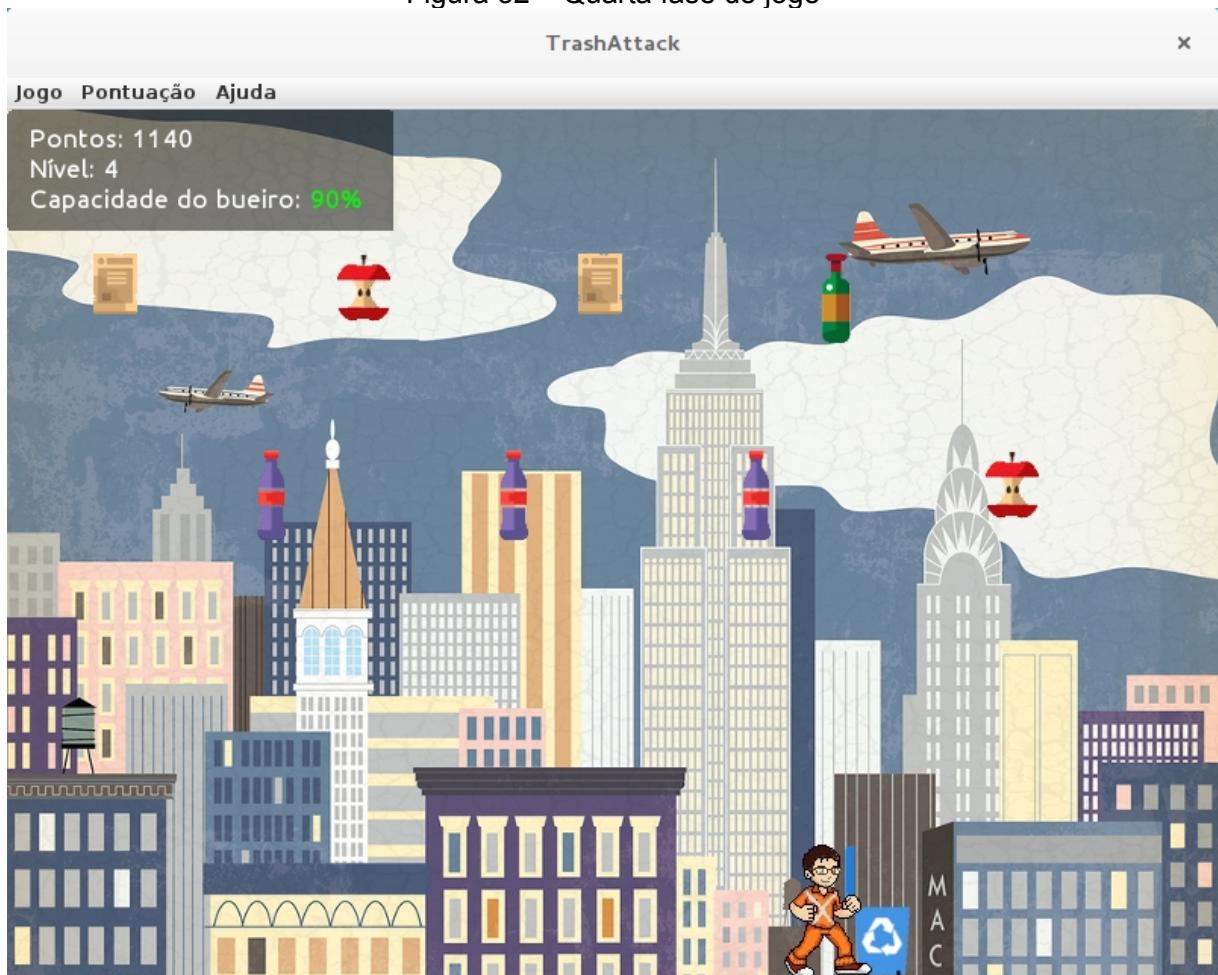
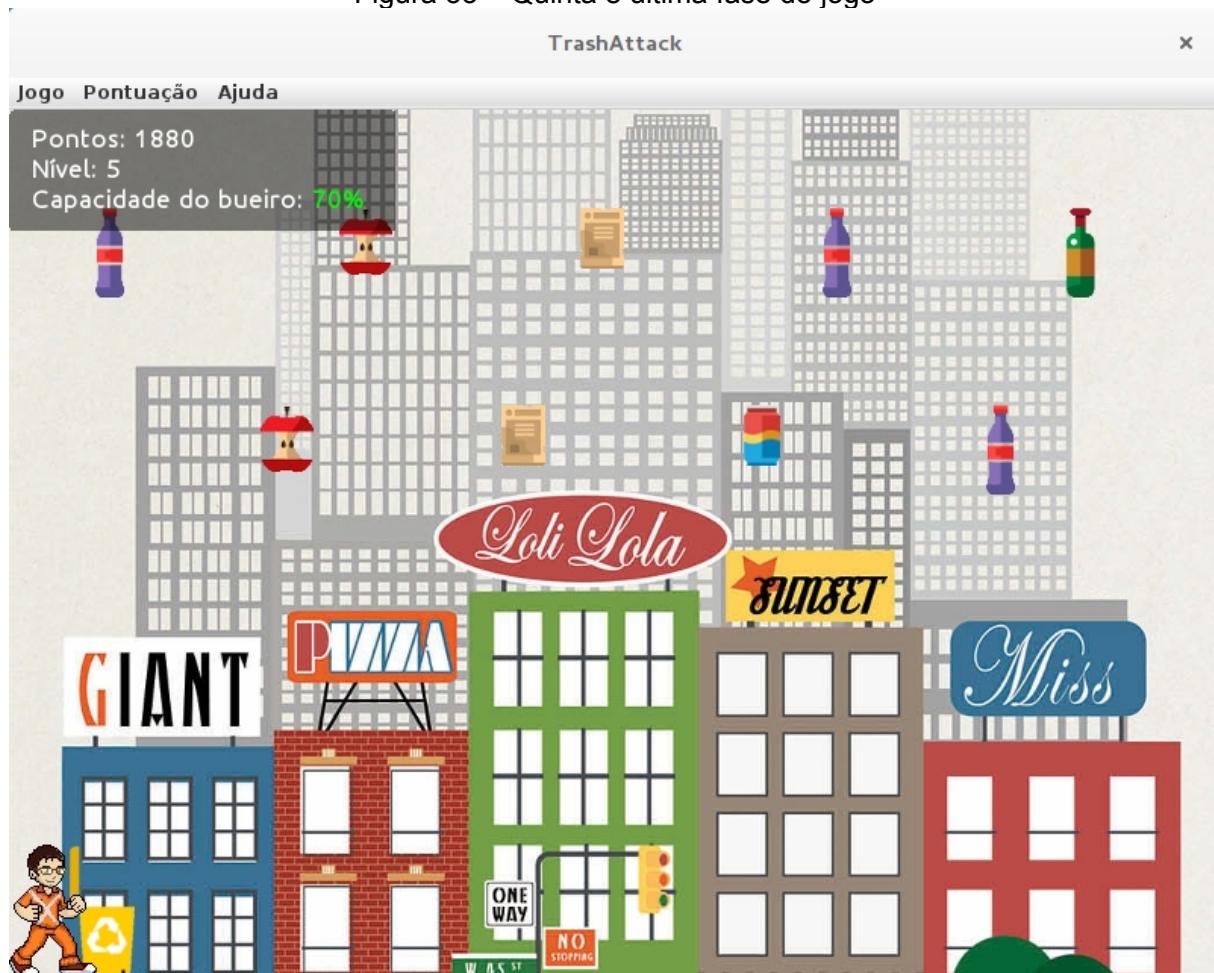


Figura 33 – Quinta e última fase do jogo



Fonte: Elaborada pelos autores

8 INTERDISCIPLINARIDADE

Todas as disciplinas referentes a este semestre desempenharam um grande papel na construção deste projeto. Algumas mais que outras, mas todas agregaram conhecimentos que foram necessários para o desenvolvimento deste jogo.

Quatro destas disciplinas são do campo das ciências exatas, contribuindo não somente para o desenvolvimento do raciocínio lógico, mas, principalmente, para a análise e resolução de problemas que envolvem cálculos. A grande maioria dos programas de computador utilizam cálculos matemáticos para executarem suas tarefas, os jogos seguem na mesma direção. Para definir as posições dos elementos na tela, detecção de colisões, sistema de pontuação, organização dos elementos na interface gráfica, controle da quantidade de inimigos por fase e suas velocidades, entre tantas outras variáveis necessárias para o bom funcionamento do jogo, necessitam de cálculos matemáticos precisos. Sendo esta a principal contribuição destas disciplinas.

Outras duas disciplinas, desta vez relacionadas a programação, tiveram incrível importância para este projeto ser finalizado com sucesso: ALPOO (Aplicação de Linguagem de Programação Orientada a Objetos) e LPBD (Linguagem de Programação de Banco de Dados). Sendo a primeira a que mais se destacou, contribuindo com a teoria e práticas necessárias para a boa estruturação de um programa e a aplicação de padrões de projetos, como o MVC e o DAO(embora não utilizado em sua integridade neste jogo).

Ter adquirido tais conhecimentos garantiu um código limpo, organizado, eficiente, reutilizável e bastante flexível. Permitindo modificar e acrescentar novas funcionalidades de maneira simples e extremamente produtiva. Sem a utilização de tais padrões, o desenvolvimento seria mais lento, improdutivo e resultando em um jogo com código fonte extramente difícil de manter e expandir.

Com o passar dos semestres, fica cada vez mais evidente a importância das disciplinas incluídas na grade curricular e como unidas formam a base de conhecimentos necessários para a realização de atividades diversas no âmbito da Ciência da Computação.

REFERÊNCIAS

KON, Fabio. **Uma visão geral de UML**. Disponível em: <<https://www.ime.usp.br/~kon/presentations/UMLIntro.pdf>>. Acesso em: 20 de novembro de 2015.

DIFFENDERFER, Philip. **Generic game loop**. Disponível em: <<http://www.gameprogblog.com/generic-game-loop/>>. Acesso em: 20 de novembro de 2015.

AGAINST THE GRAIN. **The game loop**. Disponível em: <<http://obviam.net/index.php/the-android-game-loop/>>. Acesso em: 20 de novembro de 2015.

ENTROPY INTERACTIVE. **The game loop**. Disponível em: <<http://entropyinteractive.com/2011/02/game-engine-design-the-game-loop/>>. Acesso em: 20 de novembro de 2015.

WITTERS, Koen. **deWITTERS Game loop**. Disponível em: <<http://www.koonsoolo.com/news/dewitters-gameloop/>>. Acesso em: 20 de novembro de 2015.

ABRINDO O JOGO. **Desenvolvimento de jogos digitais em Java – Índice do curso**. Disponível em: <<http://abrindoojogo.com.br/djj-index>>. Acesso em: 20 de novembro de 2015.

AGAINST THE GRAIN. **Building games using the MVC Pattern – Tutorial and introduction**. Disponível em: <<http://obviam.net/index.php/the-mvc-pattern-tutorial-building-games/>>. Acesso em: 20 de novembro de 2015.

WITTERS, Koen. **Game architecture: Model-View-Controller**. Disponível em: <<http://www.koonsoolo.com/news/model-view-controller-for-games/>>. Acesso em: 20 de novembro de 2015.

ÖLÇGEN, Atamert. **Why MVC is not an ideal model for games**. Disponível em: <http://blog.muhuk.com/2013/12/21/why_mvc_is_not_an_ideal_model_for_games.html#.VIRM7eCaGcz>. Acesso em: 20 de novembro de 2015.

BUTLER, Tom. **Model-View-Confusion part 1: The View gets its own data from the Model**. Disponível em: <<https://r.je/views-are-not-templates.html>>. Acesso em: 20 de novembro de 2015.

LAMIM, Jonathan. **MVC – O padrão de arquitetura de software**. Disponível em: <https://www.oficinadanet.com.br/artigo/1687/mvc_-_o_padrao_de_arquitetura_de_software>. Acesso em: 20 de novembro de 2015.

SILVA, Tiago F. **MVC não é sobre camadas!**. Disponível em: <<https://tiagodev.wordpress.com/2011/01/29/mvc-nao-e-sobre-camadas/>>. Acesso em: 20 de novembro de 2015.

FOWLER, Martin. **Model View Controller**. Disponível em: <<http://martinfowler.com/eaaCatalog/modelViewController.html>>. Acesso em: 20 de novembro de 2015.

FOWLER, Martin. **GUI Architectures**. Disponível em: <<http://martinfowler.com/eaaDev/uiArches.html>>. Acesso em: 20 de novembro de 2015.

MACORATTI, José Carlos. **Padrões de projeto: O modelo MVC**. Disponível em: <http://www.macoratti.net/vbn_mvc.htm>. Acesso em: 20 de novembro de 2015.

TABORDA, Sérgio Manuel Marcos. **MVC**. Disponível em: <<https://sergiotaborda.wordpress.com/desenvolvimento-de-software/java/patterns/mvc/>>. Acesso em: 20 de novembro de 2015.

MIRANDA, Fábio. **Considerações Históricas sobre o Padrão MVC**. Disponível em: <<http://fabiolnm.blogspot.com.br/2009/12/consideracoes-historicas-sobre-o-padroao.html>>. Acesso em: 20 de novembro de 2015.

MIRANDA, Fábio. **Do MVC para o MVP (Model-View-Presenter)**. Disponível em: <<http://fabiolnm.blogspot.com.br/2009/12/do-mvc-para-o-mvp-model-view-presenter.html>>. Acesso em: 20 de novembro de 2015.

STACK OVERFLOW. **MVC: should view talk with model directly?** Disponível em: <<http://stackoverflow.com/questions/18872991/mvc-should-view-talk-with-model-directly>>. Acesso em: 20 de novembro de 2015.

GUJ. **Aplicação do padrão MVC no desenvolvimento de Jogos**. Disponível em: <<http://www.guj.com.br/java/309463-aplicacao-do-padrao-mvc-no-desenvolvimento-de-jogos>>. Acesso em: 20 de novembro de 2015.

FICHA DE ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

Atividades Práticas Supervisionadas (laboratórios, atividades em biblioteca, Iniciação Científica, trabalhos individuais e em grupo, práticas de ensino e outras)

Nome: Daniela de Sales Santana

RA: C 32288-1 CURSO: CIÊNCIAS DA COMPUTAÇÃO

CAMPUS: TATUAPE' SEMESTRE: 3º Semestre TURNO: Noite

DATA	ATIVIDADE	ALUNO	TOTAL DE HORAS	ASSINATURA	PROFESSOR
20/09/2023	Ler e comentando bibliografia		00:00:00		
21/09/2023	Tema				
22/09/2023	Desenvolvimento				

TOTAL DE HORAS:

FICHA DE ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

Aulas Supervisionadas (laboratórios, atividades em biblioteca, Iniciação Científica, trabalhos individuais e em grupo, práticas de ensino e outras)

NAME: Diego Horne (depois da Síntese)

RA: C-17339-8

CURSO: Ciéncias da Computa o

CAMPUS: Totucayé

SEMESTRE: 1º

SEMESTRE: 1^º Semestre **TURNO:** Noite

卷之三

TOTAL DE HORAS: _____

FICHA DE ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

Actividades Práticas Supervisionadas (laboratórios, atividades em biblioteca, Iniciação Científica, trabalhos individuais e em grupo, práticas de ensino e outras)

NOME: Fábio Fernandes Soárez

RA: 61367415 CURSO: Licenciatura em Contabilidade
CAMPUS: Tataperu SEMESTRE: 4º Semestre TURNO: noite

TOTAL DE HORAS: _____

FICHA DE ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

Atividades Práticas Supervisionadas (laboratórios, atividades em biblioteca, Iniciação Científica, trabalhos individuais e em grupo, práticas de ensino e outras)

NOME: Wanderson Martins Oliveira

RA: C 20275-4

CAMPUS: Tatuá

CURSO: Ciéncia da Computaçao

SEMESTRE: 4º Semestre

SEMESTRE: 4º Semestre TURNO: Noite

TOTAL DE HORAS:

FICHA DE ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

atividades Práticas Supervisionadas (laboratórios, atividades em biblioteca, Iniciação Científica, trabalhos individuais e em grupo, práticas de ensino e outras)

NAME: Yury Rodrigues Anunciação

RA: C20336-0 CURSO: Ciencia da Computação

CAMPUS: Tatuvapē

SEMESTRE: 4º SEMESTRE/TURNO: Noite

ASSINATURA				
DATA	ATIVIDADE	TOTAL DE HORAS	ALUNO	PROFESSOR
	Levantamento bibliográfico		Yury R.	
	Tema		Yury R.	
	Desenvolvimento		Yury R.	

TOTAL DE HORAS: