

**UNIVERSIDADE PAULISTA**  
**Instituto de Ciências Exatas e Tecnologia – ICET**

CARLOS EDUARDO MARQUETTI CORREA DA SILVA – F33IEF7

GILBERTO DE ALMEIDA CARVALHO JÚNIOR – F202453

LAÍS LOBO TEIXEIRA – N540CA7

PEDRO HENRIQUE TERREIRO DE ARRUDA – N686650

THIAGO DE PAULA SOUZA – F33JIC8

YAN GABRIEL SILVA QUEIROZ – N625414

**DESENVOLVIMENTO SUSTENTÁVEL**

Avaliação sustentável com etiologia reciclável

Sorocaba

2021

CARLOS EDUARDO MARQUETTI CORRÊA DA SILVA – F33IEF7

GILBERTO DE ALMEIDA CARVALHO JÚNIOR - F202453

LAÍS LOBO TEIXEIRA – N540CA7

PEDRO HENRIQUE TERREIRO DE ARRUDA – N686650

THIAGO DE PAULA SOUZA – F33JIC8

YAN GABRIEL SILVA QUEIROZ – N625414

## **DESENVOLVIMENTO SUSTENTÁVEL**

Avaliação sustentável com etiologia reciclável

Trabalho da Atividade Prática Supervisionada (APS) do  
curso de Ciência da Computação apresentado à  
Universidade Paulista - UNIP

**Orientador Prof. Reverdan Almeida Springer**

Sorocaba

2021

## RESUMO

Tendo em vista que a reciclagem influencia e permite o desenvolvimento sustentável, sua utilização está focada em métodos que insistem na preservação a vida, pois se trata em seu conteúdo formas complexas que abrangem diversas áreas e/ou setores. O crescimento industrial aumentou gradativamente nos últimos anos, isto é: o aumento descontrolado da população com seu consumo de produtos que ampliam a quantidade de resíduos degradáveis e não degradáveis. No entanto, com o aumento do consumo processa-se a reciclagem que é um aspecto necessário para o equilíbrio do planeta e a saúde das pessoas. A reciclagem possui benefícios que podem ser citados até mesmo como a redução dos gases do efeito estufa – que influenciam nas mudanças climáticas – e também na preservação de fontes de matéria-prima. É necessário entender que a quantidade de lixo produzido pelas atividades humanas é um problema mundial uma vez que alguns materiais demoram centenas de anos para se decompor. Através da conscientização, será nítido a forma correta de consumo, reaproveitamento e descarte adequado, para que seja essencial edificar uma sociedade sustentável. Logo, o tema proposto possui como fundamento um jogo que ressalta como principal objetivo a conscientização dos internautas em foco infantil para retratar uma ideologia de fácil entendimento que consiga idealizar e abranger a população.

**Palavras-chave:** reciclagem, desenvolvimento sustentável, conscientização

## **ABSTRACT**

Since recycling influences and allows for sustainable development, its use is focused on methods that insist on preserving life since its content deals with complex forms that cover different areas and/or sectors. Industrial growth has gradually increased in recent years, that is: the uncontrolled increase in the population with its consumption of products increases the amount of degradable and non-degradable waste. However, with the increase in consumption, recycling is taking place, which as a necessary aspect for the balance of the planet and the health of people. Recycling has benefits that can be cited even as the reduction of greenhouse gases – which influence climate change – and also in the preservation of sources of raw materials. It is necessary to understand that the amount of waste produced by human activities is a worldwide problem since some materials take hundreds of years to decompose. Through awareness, the correct form of consumption, reuse and proper disposal will be clear, so that it is essential to build a sustainable society. Therefore, the proposed theme is based on a game that emphasizes internet users' focus on children as a main objective to portray an easily understood ideology that can idealize and reach the entire population.

**Keywords:** recycling, sustainable development, awareness

## SUMÁRIO

<b>1</b>	<b>INTRODUÇÃO .....</b>	<b>8</b>
1.1	Justificativa .....	9
1.2	Objetivos .....	9
<b>2</b>	<b>DESENVOLVIMENTO .....</b>	<b>10</b>
<b>3</b>	<b>FUNDAMENTAÇÃO TEÓRICA .....</b>	<b>13</b>
3.1	Desenvolvimento sustentável .....	13
3.1.1	Relatório brundland .....	13
3.1.2	Equilíbrio entre os três pilares sustentáveis .....	14
3.1.3	Reciclagem e aquecimento global .....	15
3.1.4	Iniciativas sustentáveis .....	16
3.1.5	Reciclagem energética .....	17
3.1.6	Reciclagem de resíduos eletrônicos .....	19
3.1.7	Melhorias que podem ser tomadas .....	20
3.1.8	Reciclagem na educação infantil .....	21
3.2	Introdução a java .....	21
3.2.1	História do jogo .....	22
<b>4</b>	<b>RELATÓRIO DO CÓDIGO FONTE .....</b>	<b>25</b>
<b>5</b>	<b>METODOLOGIA .....</b>	<b>39</b>
<b>6</b>	<b>RESULTADOS .....</b>	<b>40</b>
<b>7</b>	<b>CONSIDERAÇÕES FINAIS .....</b>	<b>41</b>
<b>8</b>	<b>REFERÊNCIAS .....</b>	<b>42</b>
<b>9</b>	<b>CÓDIGO FONTE .....</b>	<b>44</b>
<b>10</b>	<b>FICHAS DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS .....</b>	<b>95</b>

## 1. INTRODUÇÃO

O planeta está passando por diversas dificuldades ambientais que como os maiores causadores os seres humanos. A grande parte da população não demonstra interesse em buscar informações complexas relacionadas ao desequilíbrio ambiental, e isso tudo porquê esse problema ocorre em um ritmo desacelerado. Logo, acaba não atingindo diretamente a maioria da população.

Embora a grande parte da sociedade tenha em mente que não está sendo vítima do aquecimento global, ela está sendo diretamente relacionada à nossa vivência diária. Como por exemplo: o oxigênio e até mesmo os alimentos consumíveis. Os vegetais que emanam da terra possuem agrotóxicos exacerbados que fazem mal para o organismo humano e agride diretamente cada vez mais ao solo. Os animais ficam geneticamente mutantes por conta do excesso de lixo em seus habitats, as geleiras derretem devido ao aumento da temperatura, as chuvas com um enorme teor de substâncias fóssil devido a grande quantidade de carbono na atmosfera. Mas o foco principal é orientar os humanos para evitar ou propor mudanças de hábitos para a melhoria do planeta, pois tudo começa pela conscientização da situação presente.

Para fazer esse tema vir à tona novamente, será necessário uma exposição do tema em massa para que haja uma conscientização geral e de fácil entendimento. O projeto na APS é um jogo que visa demonstrar o quão prejudicial é o descarte indevido do lixo nos mares e nas ruas das grandes cidades. Como é de fácil compreensão, o projeto é voltado para um meio mais didático, que visa tratar o tema e atingir internautas no modo geral, de uma maneira simples, objetiva e divertida.

Em relação à reciclagem, o jogo conta com itens especiais misteriosos encontrados durante as fases que possibilita e instiga o jogador a querer junta-los para verificar qual será o resultado final ao juntar todos os quatros itens especiais presentes. Os quatro itens são respectivamente uma garrafa pet, uma garrafa de vidro, uma pilha e uma caixa de papelão. No fim do jogo, serão mostrados os destinos dos itens entregues a personagens que reciclam e fazem boa utilização de tais materiais. Logo, será demonstrado a sua utilidade e conseqüentemente sua fonte de renda, tornando assim uma dupla ligação em relação à reciclagem em que será vista como uma ajuda

ao meio ambiente e também às pessoas que dependem da venda dos materiais reciclados para a própria sobrevivência.

### **1.1 Justificativa**

A escolha do tema originou-se a partir de um debate do grupo sobre como os internautas podem ser influenciados (em foco sob a conscientização infantil) através da reciclagem pelo desenvolvimento sustentável.

### **1.2 Objetivos**

O objetivo central deste trabalho é ressaltar os impactos ambientais que priorizam a reciclagem e a sua influência nas mudanças climáticas, conscientizando os internautas a terem hábitos menos degradantes ao meio ambiente.

Em suma, destacam-se alternativas que transmitem costumes sustentáveis para que através do jogo criado pelos integrantes do grupo, consolide a população a gerar recomendações que possam favorecer a preservação natural. O trabalho possui como foco, o público alvo destinado ao infantil, pois o mesmo possui facilidade de acesso e entendimento para todas as idades, sendo incluso os adultos e idosos.

Através disso, focaliza de uma forma explicativa o que de fato é a reciclagem e como ela influencia na preservação a vida. Inclui, no entanto, questões ambientais que resultam em efeitos benéficos para auxiliar a sociedade a ser vista como um todo.

## 2 DESENVOLVIMENTO

Primeiramente, foi necessário criar uma classe principal chamada Game. Dentro dela, apresenta-se a principal função de trazer e obter todos os dados do jogo que corresponde uma engine (motores que reúnem informações necessárias ao desenvolvimento de jogos). Na grande parte das classes principais, sempre haverá um método chamado Tick que coincide com a lógica de trás de uma classe. Além dessa lógica de dados, encontra-se outro método render que visa a renderização dos gráficos do jogo.

Para dar continuidade as classes, vale ressaltar que existe uma classe chamada spritesheet que possui como funcionamento a definição de todos os valores básicos e principais de quaisquer imagens presentes no código. Logo após, foi criado a parte do jogador. Ela serve como uma definição ampla de frames da animação, quantidade máxima de sprite utilizada, pulo, etceteras. Além disso, a mesma armazena partes referentes ao Protagonista que seta colisões com blocos e monstros. Isso evita que o personagem não atravessasse. implementando um sistema de colisão pra ficar fixo ao piso. Se não, iria ocorrer um erro e o personagem passaria por ela. O gráfico do player renderiza sprites específicas do mesmo. Todo o restante do objeto da classe spritesheet - com exceção do bloco que se encontra em outra classe – possui uma classe única e distinta. Vale ressaltar que existe uma classe chamada user interface, sendo considerada a parte visual do usuário. Ela é responsável em demonstrar a verificação a vida e a quantidade de itens coletados e inimigos executados.

A câmera, por sua vez, também pode ser citada como acompanhamento. No geral, ele gera uma limitação ao mapa, tendo variáveis de coordenada da câmera que é utilizada em outras classes como a do player. Essa limitação seta o tamanho máximo e o mínimo para que não ultrapasse o tamanho concreto e não ocorra problemas de bugs na câmera. O gráfico do Player renderiza sprites específicas do player.

Na classe entidade ou entity - todas as entidades possíveis –, será pegado todos os sprites, tais como itens coletáveis, inimigos e o próprio jogador. Esse método de entidade realiza e define todos os atributos que as outras classes utilizam; também vale ser citado que ela pode ser dita como parâmetro. Os sprites salvam os dados em toda variável array (ele salva diversos sprites em vários arrays diferentes, salvando a



seguir as sprites da entidade em uma array específico da entidade) que é utilizada para serem animadas posteriormente. Dentro da classe entidade, existe um método chamado Comparator que têm como função comparar duas entidades e organiza-los em um ArrayList. Para verificar se existe colisão nas entidades será necessário utilizar uma função chamada iscoliding. Ao zerar a vida e mandar a mensagem do game-Over, será demonstrado nessa classe. No entanto, a verificação que zera a vida encontra-se na classe player que manda a mensagem para a classe gamer executar o código do game-over.

Enemy – herdado da classe entidade – cria um construtor (ligação) e retira todas as informações dos monstros. O inimigo move-se somente na esquerda e direita diante da mesma. Ele seta a seguir suas características e criação. Após, será determinado uma velocidade, vida, entre outros, que abrange e define a lógica utilizada pelos monstros. Outro fato que pode ser citado é um método de classe que essa herança possui. A mesma renderiza a sua forma.

Na classe world, a função principal será pegar os atributos como os tamanhos de largura e altura. Também define quais são os tamanhos dos pisos e quais são para conseguir renderizar os blocos. Além de validar os pisos, ele válida por cores os sprites das entidades. Cada componente diante disso possui um número de série que é utilizado para definir a parte do código feita para aquela entidade específica, isto é: de acordo com a cor hexadecimal na imagem do mapa, os if e else if's farão a validação dos pixels de acordo com o código HEX das cores.

Para evitar repetições do código, foi criada uma classe em chamada tile. Essa classe tem como função o armazenamento de todas as sprites dos blocos. Através dela, será usado métodos em que dividem essas mesmas sprites com o intuito de separá-los e defini-los em suas classes “filhas” (flor, sand, etecetera).

Inicialmente, foi criada uma classe moeda no jogo que tinha como objetivo guardar os frames da animação e outras informações herdadas da classe entidade. No decorrer do jogo, essa classe foi modificada para cinco classes diferentes: paper, plastic, battery, glass e cardboard. A mudança ocorreu para fazer mais sentido ao game pois abrange o conceito de reciclagem. Foi criado um sexto item (steregg) em

segredo que possui como fundamental levar o jogador a fase final, sendo isso o mesmo pula metade da primeira e segunda fase.

A classe sound foi criada a fim de definir o som. Em início, foi realizado um construtor que recebe dois parâmetros. Dentro disso, é realizada a instância thread que têm como funcionamento a lógica para tentar realizar o input do som. O segundo parâmetro é uma condição chamada if. A mesma coloca o som em um looping para que o jogo tenha uma música constante.

Em relação ao menu, foi proposto algo mais padronizado em que se encontra as opções de start e exit. Quando pressionadas, elas enviam uma orientação lógica ao código para executar uma função específica de entrar ao jogo ou sair do mesmo.

Para finalizar, foi criada uma classe background: ela pega a imagem e seta o tamanho e o início da mesma – sendo que as imagens citadas são salvas em imagens próprias nas pastas res e bin do jogo. Ela é dita como uma imagem estática no fundo do jogo.

### **3 FUNDAMENTAÇÃO TEÓRICA**

#### **3.1 Desenvolvimento sustentável**

O desenvolvimento sustentável surgiu na década de 1980. O termo emergiu da relação entre a preservação ambiental, sem comprometer a capacidade de atender as necessidades futuras. Nas últimas décadas, o desenvolvimento econômico foi relacionado por fatores financeiros no mercado, sem levar em consideração o fator ambiental. Logo, pode-se dizer que o desenvolvimento sustentável é tido como aquele que garante o crescimento econômico.

Quase todas as definições que descrevem o desenvolvimento sustentável possuem como base os princípios da sustentabilidade. Em suma, sustentabilidade é derivada de sustentar, apoiar, cuidar e conservar. Isto é: é a capacidade de conservação ou sustentação de um processo e/ou sistema.

Diante disso, a sustentabilidade é vista em dois níveis diferentes: sustentabilidade fraca ou forte. A fraca é a que desenvolve apenas o que foi consumido, e a forte é relaciona o consumo com a manutenção de recursos naturais. Logo, a sustentabilidade forte é menos confiante em relação ao desenvolvimento sustentável pois a mesma defende que os recursos naturais devem ser mantidos e ampliados, e não substituído.

##### **3.1.1 Relatório Brundland**

O relatório explica um termo simplório: o desenvolvimento satisfaz as necessidades do presente e não compromete a capacidade das gerações futuras de realizarem as suas próprias necessidades.

Foi elaborado através da Comissão Mundial sobre o Meio Ambiente e Desenvolvimento, apontando uma certa incompatibilidade entre desenvolvimento sustentável e os padrões de produção e consumo. Esse mesmo relatório enfatiza os problemas ambientais, tais como aquecimento global e a destruição da camada de ozônio. O próprio expressou de forma ampla a preocupação em relação ao fato de a velocidade das mudanças estarem exercendo a capacidade de propor novas soluções.

As metas propostas diante desse relatório enfatiza o banimento de guerras, adota uma proteção dos ecossistemas supranacionais – antártica e oceanos -,

implementa um programa sustentável pela Organização das Nações Unidas (ONU) que possui como medida alternativa a reciclagem de materiais reaproveitáveis entre outros.

### 3.1.2 Equilíbrio entre os três pilares sustentáveis

Com o foco em sustentabilidade, será necessário entender que a mesma engloba três conceitos: econômico, social e ambiental. A sustentabilidade ambiental, por sua vez, é o uso consciente dos recursos naturais. O termo surgiu com o objetivo de aumentar as práticas e ações que não agredem tanto ao meio ambiente, tendo em consideração a qualidade de vida. Esse tipo de sustentabilidade é focado no desmatamento, queimas e poluições, pois esses fatores são resultados de ações de humanos. Deve-se notar que pode obter melhorias através da separação de lixo, como a reciclagem.

A sustentabilidade econômica desenvolve-se através de empresas. Ela é relacionada com produção e crescimento de consumo de bens e serviços. Logo, gerar lucros e empregos em forma de conjunto de práticas administrativas e econômicas visam a preservação a vida. Algumas práticas podem ser utilizadas, como por exemplo o tratamento de resíduos orgânicos e a energia limpa.

A sustentabilidade social está ligada diretamente a um conjunto de pessoas que visam principalmente suas ações. A diminuição da desigualdade social e a garantia de serviços como saúde e educação é vista como uma forma de que as ações sociais estão funcionando.

Imagem 1 – Tripé da Sustentabilidade



Fonte: Meio Sustentável, 2019.

### **3.1.3 Reciclagem e aquecimento global**

Desde a primeira revolução industrial, percebeu-se o aumento gradativo na emissão de gases poluentes na atmosfera, sendo tido como consequência da industrialização dos processos e produtos. O consumo desses itens está interligado com a falta de conhecimento sobre os resíduos que contribuem para uma cultura de produção de lixos.

Necessariamente, é de suma importância possuir uma perceptiva repleta em relação ao desperdício de recursos naturais. Ao evitar o desperdício, será possível a preservação a vida. O lixo, por sua vez, possui ligação direta com as mudanças climáticas e o aquecimento global - processo que sofre as mais variadas consequências com o ecossistema -. Logo, será necessário compreender que a correta reciclagem impede o aquecimento, previamente que a mesma evita a realização de uma parte extremamente significativa dos processos de produção. Na sociedade, o lixo é considerado um material sem valor e por isso acaba sendo descartado. O descaso gradativo do mesmo eleva um desperdício, ocorrendo assim problemas ambientais e econômicos. Contudo, o termo lixo pode ser referido a resíduos que são capazes de reaproveitamento, por exemplo: o resíduo sólido é apontado como um material descartado resultante das atividades humanas; esse mesmo resíduo pode ser aproveitado em novos processos produtivos.

No Brasil, existe uma legislação que coordena a gestão do lixo. Segundo a Associação Brasileira de Empresas de Limpeza Pública e Resíduos Especiais (Abrelpe), em 2018 no país, foi gerado cerca de setenta e nove milhões de toneladas de lixo. Esse número enfatiza o Brasil em primeiro lugar dentre os maiores produtores de lixo na América Latina. A quantidade significativa é resultado da busca por um desenvolvimento econômico acelerado e o aumento no consumo de produtos industrializados.

No entanto, é necessário entender o quanto uma gestão correta de resíduos sólidos iria impactar positivamente a sociedade e o meio ambiente. Esses resíduos podem bloquear rios e sistemas de drenagem, que gera por sua vez inundações contribuintes para a propagação de doenças em comunidades. De acordo com a Associação Brasileira de Engenharia Sanitária Ambiental (ABES), nos primeiros três meses de 2020 cerca de 40 mil pessoas foram internadas por falta de saneamento

e/ou má gestão do lixo. Os impactos afetam o clima global pois existem determinados resíduos que emitem gases do efeito estufa, sendo assim: a decomposição da matéria orgânica que libera o gás metano, enquanto a queima descontrolada de certos materiais produz dióxido de carbono, óxido nitroso e hexafluoreto de enxofre.

Diante de um estudo feito sobre o Departamento de Economia do Sindicato Nacional das Empresas de Limpeza Urbana (Selurb), foi realizado um estudo baseado no descarte do lixo no Brasil e a queima irregular de resíduos. O estudo cita sobre o impacto dos lixões no país para a poluição atmosférica, isto é: a má gestão do lixo como responsável por emitir cerca de seis milhões de toneladas de dióxido de carbono na atmosfera. Logo, as consequências através do aquecimento têm aumentado gradativamente a temperatura média global, sendo 0,9°C.

Esse aumento progressivo da temperatura causa modificações no ecossistema, entre elas tempestades, inundações, aquecimento das águas, etecetera. Em 2019, o Painel Intergovernamental sobre Mudanças Climáticas (IPCC) divulgou um relatório que possui previsões futuras sobre essas mudanças. Nesse relatório, é citado uma variação de até um metro no nível do mar em até 2100, caso não seja feito nada. Isso retorna em desaparecimento de cidades e/ou ilhas povoadas ou muito próximas às costas – onde o mar e o solo se encontram -.

Para contribuir de uma forma positiva, é necessário estimular ideias que têm como foco o reaproveitamento dos materiais após o consumo. Será necessário enxergar o lixo como um elemento reaproveitável, sendo ditas estratégias que minimizam a geração seguindo de iniciativas sustentáveis.

#### **3.1.4 Iniciativas sustentáveis**

A conscientização instiga as pessoas a darem os primeiros passos no processo tido como mudança de uma sociedade. A partir de um ponto, cabe a cada indivíduo modificar de dentro para fora. Logo, será necessário colocar em prática o que se aprendeu durante o processo. Em suma, será algo progressivo que possuirá um certo atraso em aparecer, pois a grande parte da sociedade não irá aderir novos hábitos para tornar a vida no planeta Terra mais sustentável.

Será viável que cabe a cada indivíduo dar o exemplo próximo de suas boas práticas com o meio ambiente para que posteriormente as boas práticas sejam passadas futuramente para novas gerações, que visa a vida do planeta de uma forma mais natural e não artificial como está atualmente.

A forma mais difícil de começar será pelo primeiro passo, em que tudo fica mais natural e acaba virando parte da vida cotidiana. Será necessário dar início através da coleta de lixo e descarte do mesmo de modo apropriado – apenas em latões de lixo de materiais específicos ou convencional. Andar com sacolinhas ou algum tipo de recipiente na bagagem diária também é dita como um bom hábito, pois a mesma facilita o descarte adequado do resíduo.

Vale ressaltar que o uso de transporte público e bicicletas auxiliam para obter uma qualidade de vida melhor e também age de acordo a reduzir a emissão de gases fósseis na atmosfera. Participar de programas que instigam doações de óleo de cozinha e outros materiais que podem ser reutilizados, ocasiona de uma forma complexa a diminuição de alguns recursos naturais desnecessários.

A preservação do meio ambiente começa com pequenas atitudes diárias, que fazem toda a diferença. Uma das mais importantes é a reciclagem do lixo.  
(ALVES, Natalia. Jun, 2015)

### **3.1.5 Reciclagem energética**

Sendo tida como um processo de aproveitamento energético, essa forma de reciclagem abrange de uma forma tecnológica a transformação de resíduos em energia termina e/ou elétrica. A maior parte dos resíduos sólidos que não podem ser reutilizados são utilizados nesse processo de reciclagem pois os mesmos promovem a combustão – reação química que um combustível (oxidável) reage com um comburente (material gasoso que contém o gás oxigênio), e por fim realiza o processo exotérmico -.

Os resíduos que podem ser utilizados nesse processo são os restos de alimentos, materiais higiênicos descartáveis, plásticos, etcetera. Em suma, o material mais indicado é o plástico, pois o mesmo deriva de petróleo que possui um elevado poder calorífico – quantidade de energia interna armazenada de uma determinada substância -, o que facilita a utilização na produção energética.

A energia elétrica e/ou térmica é obtida através da utilização do vapor resultante de resíduos carbonizados. Esse mesmo vapor gera um movimento chamado de energia cinética, pois ele estimula o eixo dito como turbina. Em relação aos plásticos, é produzido cerca de 650kWh de energia em cada tonelada desse material, justamente porque o movimento giratório produzido pelo eixo da bobina altera o fluxo do campo magnético dentro do gerador. Esse desempenho produz e energia elétrica.

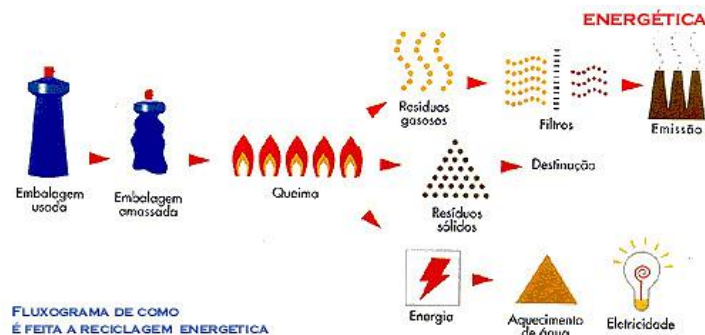
Em relação a decomposição, é válido ressaltar que os plásticos se decompõem numa temperatura elevada a 950°C. Logo, a oxidação dos gases dessa combustão ocorre por aproximadamente dois segundos, e as suas cinzas podem ser utilizadas em construções.

Para dar origem ao assunto, as primeiras usinas que utilizam essa forma de reciclagem deram-se origem em 1980. Atualmente, esse tipo de tecnologia encontra-se presente em aproximadamente trinta países. Na Alemanha, por exemplo, foram extintos os aterros sanitários (obra que garante a disposição correta dos resíduos sólidos urbanos) para dar lugar às usinas que contêm esse processo de reciclagem.

Uma das vantagens dessa prática é o método de higienização, pois a mesma elimina agentes biológicos que tendem a ser nocivos à saúde. Em relação as desvantagens, é válido ressaltar que o processo vale muito dinheiro, e por esse motivo, essa forma só pode ser aplicada quando os outros tipos não forem viáveis.

No Brasil, a implementação de usinas torna-se inviável por conta de seu valor. No entanto, a única usina brasileira existente no país é a Usina Verde, localizada no campus da Universidade Federal do Rio de Janeiro (UFRJ).

IMG 2 – Reciclagem Energética



Fonte: Reciclagem de Plásticos, 2006.



### **3.1.6 Reciclagem de resíduos eletrônicos**

Além da reciclagem dos materiais convencionais mais conhecidos como produtos orgânicos e inorgânicos, existe a reciclagem de pilhas e baterias que basicamente é composta/integrada por seis processos.

O primeiro deles é basicamente o processo de conscientização que as pessoas necessitam ter, para entenderem que os materiais eletrônicos não devem, de maneira nenhuma, serem descartados em locais inapropriados, devido aos riscos que oferecem à saúde humana caso tenham seus compostos expostos. Com a conscientização, além do descarte consciente, o processo de educar as novas gerações à também descartarem conscientemente acaba acontecendo automaticamente.

No processo seguinte encontra-se o transporte, onde as lojas que geralmente são as receptoras de todas as pilhas e baterias, recolhem o material e notificam a Green Eletron (gestora do transporte para a reciclagem), que o material está pronto para ser recolhido e ser destinado para a triagem, onde serão então recicladas e capacitadas para que sejam devidamente homologadas. Quando os materiais chegam à empresa recicladora, todos passam pelo processo de trituração, e entre os materiais existem: pilhas de lítio recarregáveis, íon e zinco, recicladas separadamente; pilhas comuns e alcalinas (AAs ou AAAs), geralmente utilizadas geralmente em aparelhos domésticos. Vale ressaltar que é importante não investir em pilhas de marcas duvidosas ou piratas, pois os componentes de sua composição podem ser totalmente tóxicos e geralmente não possuem a possibilidade de reciclagem.

As pilhas e baterias então são submetidas aos processos de reação química, onde são recuperados os sais e óxidos metálicos utilizados como as matérias primas nos processos industriais. Os componentes então são colocados em forno industrial, onde se submetem à altas temperaturas para a separação do zinco. Logo, o mineral entra em processo de recuperação em sua forma metálica e pode ser reutilizado para a fabricação de novas pilhas e baterias.

O processo de reciclagem das pilhas e baterias, além de se transformarem em materiais novos, reaproveitam os componentes químicos e metais, ajudam na

destinação dos restos e trejeitos a serem descartados sem que haja ações prejudiciais ao meio ambiente.

Imagem 3 – Reciclagem Eletrônica.



Fonte: Codel Reciclagens.

### 3.1.7 Melhorias que possam ser tomadas

Segundo dados de 2018-2020 da revista galileu e do site larplasticos, aproximadamente 50% do lixo que é gerado no país é descartado de uma forma incorreta. Uma das justificativas geradas por esses dados é a falta de financiamento, que por sua vez têm dificuldades em implantar uma política concreta capaz de realizar um descarte adequado.

Ainda na revista Galileu, foi divulgado dado das áreas do país com melhores níveis de descarte: o sul descarta cerca de 88,5% do lixo destinado de forma correta, o sudeste se encontra no segundo lugar com 51,1% de destinação correta dos resíduos. O norte e o centro-oeste possuem uma taxa de descarte adequado de respectivamente 14,1% e 14,4%. A região nordeste do país consta com um índice de 11,4%. No requisito de reciclagem, o Brasil é o mais atrasado com uma média viável em 3,7% (dados de 2018). Com os dados citados, pode-se notar que é necessário ocorrer melhorias nesse setor, pois esses eventos prejudicam o país em muitos pontos negativos.

### **3.1.8 Reciclagem na educação infantil**

A reciclagem não se começa diante da vida adulta. Ela pode ser iniciada ao ensinar as crianças a criarem bons hábitos de reciclar e ajudar o mundo a ser um lugar melhor para se viver. A reciclagem não é apenas o ato maçante de separar o lixo, existem métodos e brincadeiras para conseguir adentrar a ideia reciclar as crianças.

A nova geração está com a chave do planeta em suas mãos e cabe aos adultos dar o caminho a eles. A reciclagem não ajuda somente ao meio ambiente, mas ajuda a diminuir o desemprego, a pobreza, a fome e entre outros malefícios que o mundo enfrenta.

As escolas podem criar campanhas com as crianças e incentivar atividades dinâmicas e divertidas para que as crianças criem gosto em reciclar. As igrejas também poderiam ajudar nesse processo, e não somente alguns lugares em específico. Sempre será de suma importância dar bons exemplos para as crianças, principalmente os pais que possuem uma maior influencia na vida das mesmas.

Para começar com esse processo, será necessário influenciar as crianças através de brincadeiras, como por exemplo: montar um brinquedo de resíduo com elas e até mesmo um concurso de desenho para desenvolver o lado competitivo ao mesmo tempo de estar passando a ideia de reciclagem. Somente pequenos atos foram citados, e com isso vale ressaltar que são de fáceis entendimento e concretização. Existem diversos métodos para se concretizar a fixação do tema.

### **3.2 Introdução a java**

A criação da programação orientada a objetos dita como java deu-se início em 1990 e foi criada originalmente em 1991, quando Patrick Naughton, Mike Sheridan e James Gosling criaram um projeto na Sun Microsystems (empresa desenvolvedora de tecnologia) com o objetivo de desenvolver a “próxima onda” que aconteceria na área de informática e programação. Em 1995 foi lançado o java, que visa uma versão atualizada do Oak – que possui como significado carvalho - para a internet.

Com diversas utilizações, o java ficou bastante conhecido e em pouco tempo e ganhou o suporte de inúmeras empresas (os produtos dessa empresa teriam seus apps rodando em java), com seu sucesso em 2003. Tinha em média três milhões de

desenvolvedores. Em 2002, em seu primeiro suporte jogos mobile só era possível a criação de jogos com a capacitação de 30Kb. Com isso, o java realizou algumas criações, dentre elas estão Steve Jackson's Socery e Alien Fish Exchange.

Atualmente, o ambiente integrado a java é bem adequado para as pessoas que possuem interesse em desenvolvimento de jogos, com grandes exemplos em seu nome, como Minecraft e até mesmo Pokémon em realidade aumentada (o mesmo possui parceria entre a Niantic e a Nintendo que fizeram esse jogo). Nesse jogo, eles utilizaram um Framework (conjunto de bibliotecas e recursos de programação) chamado LibGDX, que permite aos desenvolvedores a criação de jogos em multiplataformas.

Por sua vez, o java possui um back-end extremamente robusto que dá uma liberdade de criação ao desenvolvedor. Em uma visão mais geral sobre a linguagem, a mesma é uma das mais utilizadas no mundo, sendo considerada bastante versátil para a criação de qualquer programa – seja ele um jogo, aplicativo, etcetera. A linguagem possui multi-threades (operações múltiplas que podem ocorrer ao mesmo tempo) que facilitam bastante o trabalho do programador.

Por possuir uma grande comunidade, existe atualmente milhares de informações disponíveis em fóruns, sites, grupos, redes sociais, etcetera. As mesmas têm como fundamentação um bom suporte por conta da troca de informações. Dentre todas essas vantagens, ele possui falhas, dois exemplos disso seriam os processamentos numéricos e a pré-compilação. O processamento numérico possui demandas rígidas quanto ao bom funcionamento dos seus tipos numéricos, ele acaba tornando-se um pouco lento. A pré-compilação varia da capacidade do computador pessoal, e os processos como o citado acima de torna lento no Ambiente de Desenvolvimento Integrado (IDE).

You don't want to reinvent the wheel since Java is about abstraction.  
(ANDREW, Davison. Killer Game Programming, 2005)

### **3.2.1 História do jogo**

Para abertura ao tópico, será necessário ressaltar que o foco de estilo do jogo é situado em plataforma 2D. No parágrafo a seguir, será citado um pequeno capítulo referente ao jogo que visa influenciar os jogadores a praticarem a coleta de

reciclagens; o jogo possui um cunho educativo em que o protagonista pratica hábitos como a reciclagem a fim de tornar o mundo um lugar melhor. É evidente que o cenário consta elementos tido como tóxicos.

“Devastado com a situação presente no mundo atual citada na televisão, Derick encaminha-se até a praia com a intenção de coletar reciclagens. Ao chegar à mesma, o garoto depara-se com lixos flutuantes que lembram uma toxidade-viva. É uma consequência dos atos de humanos. Os lixos flutuavam e demonstravam a sua forma como algo extremamente violento. E a própria criação disso tudo, diante da percepção do garoto, eram os humanos. Os próprios estavam congelados com um vislumbre épico e perspicaz do agora pairando diante de murmúrios sobre coisas difíceis de compreender. Os gritos, o desespero... tudo se passava diante dos olhos de Derick que se encontrava extremamente aflito.

Logo, o menino teve uma ideia: ‘por que não coletar o pouco que resta aqui? Quem sabe isso torne a situação presente melhor’. E então, para dar início, o garoto começou a coletar o que pudera observar na praia. Esquivando-se dos monstros, o mesmo encaminhou-se até a cidade e realizou o mesmo procedimento. Em pouco tempo, notava-se que os monstros estavam diminuindo. Ao prosseguir com a situação, Derick percebeu que o que ele tinha para acarretar, ele conseguiu naquele momento. Só que era necessária a ajuda de outras pessoas para que o processo torne-se maior. No entanto, ele iria continuar com a coleta de reciclagem, pois o mais importante é pensar no bem-estar e preservação à vida.”

Imagem 4 – Primeira Fase



Fonte: IDE Eclipse

Imagem 5 – Segunda Fase



Fonte: IDE Eclipse

Imagem 6 – Terceira Fase



Fonte: IDE Eclipse

## 4 RELATÓRIO DO CÓDIGO FONTE

Para uma melhor informação foi destinada um tempo de criação para este tópico que tem a função de trazer um relatório desde o início até o fim da programação do código.

Antes de tudo, será necessário criar a engine dita como motor do jogo. Para isso, será necessário ter o construtor “game” que obtém diversos valores de classes externas.

Imagem 7 – Classe Game

```
public Game() {
    addKeyListener(this);
    addMouseListener(this);
    addMouseMotionListener(this);

    // comando para definir o tamanho da janela
    setPreferredSize(new Dimension(WIDTH * SCALE, HEIGHT * SCALE));

    initFrame();
    image = new BufferedImage(WIDTH, HEIGHT, BufferedImage.TYPE_INT_RGB);

    // comandos para iniciar as sprites
    spritesheet = new Spritesheet("/spritesheet.png");
    background1 = new Spritesheet("/level1BCK.png");
    background2 = new Spritesheet("/level2BCK.png");
    background3 = new Spritesheet("/level3BCK.png");
    backgroundMENU = new Spritesheet("/backgroundMENU.png");
    entities = new ArrayList<Entity>();
    player = new Player(WIDTH / 2 - 30, HEIGHT / 2, 10, 10, 1.4, ENTITY.PLAYER_SPRITE_RIGHT[0]);

    world = new World("/level" + CUR_LEVEL + ".png");
    ui = new UI();

    lightMap = new int[WIDTH * HEIGHT];
    pixels = ((DataBufferInt) image.getRaster().getDataBuffer()).getData();
    menu = new Menu(); // chama a classe menu

    soundtrack = new Sound("/sound/country banjo", true);

    // comandos para inicializar as fontes
    try {
        newFont = Font.createFont(Font.TRUETYPE_FONT, stream).deriveFont(45f);
        newFont2 = Font.createFont(Font.TRUETYPE_FONT, stream2).deriveFont(60f);
        newFont3 = Font.createFont(Font.TRUETYPE_FONT, stream3).deriveFont(150f);
    } catch (FontFormatException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }

    entities.add(player);
}
```

Fonte: IDE Eclipse

Método em que foi escolhido todos os valores dos atributos para inicialização e finalização da janela do jogo.

Imagem 8 – Inicialização e Finalização da Janela

```
// Comandos de configuração da janela
public void initFrame() {
    frame = new JFrame("Plataforma Reciclagem");
    frame.add(this);
    frame.setResizable(false);
    frame.pack();
    frame.setLocationRelativeTo(null);

    // Comando q finaliza o programa sem deixar ele rodando fechado
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
    frame.setVisible(true);
}
}
```

Fonte: IDE Eclipse

Métodos que além de inicializar finaliza o jogo e todo o seu conteúdo do mesmo.

Imagem 9 – Finalização do Jogo

```
// Metodo de start do jogo
public synchronized void start() {
    thread = new Thread(this);
    isRunning = true;
    thread.start();
}

// Metodo de stop do jogo
public synchronized void stop() {
    isRunning = false;
    try {
        thread.join();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}
```

Fonte: IDE Eclipse

Método que controla todas as atualizações no jogo, como valores das entidades ou troca de fases.



Imagem 10 – Renderização de todos os gráficos

```
// Método de renderização dos gráficos
public void render() {
    // Código que não precisa para retornar o gráfico do jogo
    BufferStrategy bs = this.getBufferStrategy();
    if (bs == null) {
        this.createBufferStrategy(3);
        return;
    }
    Graphics g = image.getGraphics();
    g.setColor(new Color(127, 127, 255));
    g.fillRect(0, 0, WIDTH, HEIGHT);
    if (CUR_LEVEL == 1) g.drawImage(backgrounds.BC1, -Camera.x, -Camera.y, 3840, 160, null);
    else if (CUR_LEVEL == 2) g.drawImage(backgrounds.BC2, -Camera.x, -Camera.y, 3840, 160, null);
    else if (CUR_LEVEL == 3) g.drawImage(backgrounds.BC3, -Camera.x, -Camera.y, 312, 268, null);
    // Comando para renderizar o jogo
    world.render(g);
    Collections.sort(entities, entity.nodesorter);
    for (int i = 0; i < entities.size(); i++) {
        Entity e = entities.get(i);
        e.render(g);
    }
    g.dispose();
    bs = bs.getDrawGraphics();
    g.drawImage(image, 0, 0, WIDTH * SCALE, HEIGHT * SCALE, null);
    ui.render(g);
    // IF para mostrar menu
    if (GameState == "MENU") menu.render(g);
    // IF para mostrar o game over e RESERVAR o game over e RESERVAR o JOGO
    if (GameState == "GAME_OVER") {
        Graphics g2 = (Graphics) g;
        // Comando para mostrar o game over
        g.setFont(new Font("serif", Font.PLAIN, 16));
        g.setColor(new Color(0, 0, 0, 255));
        g.fillRect(0, 0, WIDTH * SCALE, HEIGHT * SCALE);
        g.setColor(Color.red);
        g.drawString("Game Over", (WIDTH * SCALE) - 150, 210);
        g.setFont(new Font("serif", Font.PLAIN, 12));
        g.setColor(Color.red);
        // IF para mostrar o "Pressione Enter..."
        if (showMessageGameOver) g.drawString("Pressione Enter para reiniciar o jogo", (WIDTH * SCALE) - 600, 260);
        // Comando para mostrar o level
        if (restart == true) {
            GameState = "MENU";
            entities.clear();
            entities.add(player);
            world = new World("level" + CUR_LEVEL + ".png");
            Player.currentCoins = 0;
            Player.life = 1;
            Player.maxCoins = Player.maxCoins / 2;
            Player.currentEnemies = 0;
            Player.maxEnemies = Player.maxEnemies / 2;
            Game.player.updateCamera();
        }
    }
}
bs.show();
}
```

Fonte: IDE Eclipse

Método que controla a taxa de atualização do jogo, além de limitar o FPS.

Imagem 11 – Taxa de Atualização

```
// Método de controle do loop do jogo
public void run() {
    long lastTime = System.nanoTime();
    double amountOfTicks = 60.0; // FPS
    // Variável que vai ser usada para saber o tempo que o jogo leva a fazer um update
    double ns = 1000000000 / amountOfTicks;
    double delta = 0;
    int frames = 0;
    double timer = System.currentTimeMillis();
    requestFocus();
    while (isRunning) {
        long now = System.nanoTime();
        delta += (now - lastTime) / ns;
        lastTime = now;
        // IF que controla o FPS
        if (delta >= 1) {
            tick();
            render();
            frames++;
            delta--;
        }
        // IF para mostrar no console a quantidade de ticks e a quantidade de FPS
        // Informar na variável amountOfTicks
        if (System.currentTimeMillis() - timer >= 1000) {
            System.out.println("FPS: " + frames);
            frames = 0;
            timer += 1000;
        }
    }
    stop();
}
```

Fonte: IDE Eclipse

Método que controla o próprio jogador e seus controles como pulo, andar para direita, etcetera.

Imagem 12 – Método de Controlar

```
// Controlar para controlar o jogador
@Override
public void keyPressed(KeyEvent e) {
    if (e.getKeyCode() == KeyEvent.VK_RIGHT) {
        player.right = true;
    } else if (e.getKeyCode() == KeyEvent.VK_LEFT) {
        player.left = true;
    }

    // Para quando o jogador pressiona a tecla de espaço para pular
    // Quando
    if (e.getKeyCode() == KeyEvent.VK_SPACE) {
        player.jump = true;
    }

    // MENU - up
    if (e.getKeyCode() == KeyEvent.VK_UP) {
        if (GameState == "MENU") {
            menu.UP = true;
        }
    }
    // MENU - down
    else if (e.getKeyCode() == KeyEvent.VK_DOWN) {
        if (GameState == "MENU") {
            menu.DOWN = true;
        }
    }

    // Enter do principal
    if (e.getKeyCode() == KeyEvent.VK_ENTER) {
        restart = true;
        // MENU - ENTER
        if (GameState == "MENU") {
            menu.OK = true;
        }
    }
}

// Controlar para detectar quando o jogador pressiona a tecla de espaço para pular
@Override
public void keyReleased(KeyEvent e) {
    if (e.getKeyCode() == KeyEvent.VK_RIGHT) {
        player.right = false;
    } else if (e.getKeyCode() == KeyEvent.VK_LEFT) {
        player.left = false;
    }
}
}
```

Fonte: IDE Eclipse

A classe spritesheet possui a função de obter as imagens de um arquivo externa, ela também provê os gráficos para todas as outras classes do aplicativo Java.

Imagem 13 – Classe Spritesheet

```
public class Spritesheet {
    private BufferedImage spritesheet;

    public Spritesheet(String path) {
        try {
            //Variável que lida com a imagem de um arquivo para a
            spritesheet = ImageIO.read(getClass().getResource(path));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    //Método que retorna a imagem de acordo com as coordenadas e altura
    public BufferedImage getSprite(int x, int y, int width, int height) {
        return spritesheet.getSubimage(x, y, width, height);
    }
}
```

Fonte: IDE Eclipse

Esse método controla toda física do jogador, além de centralizar a câmera do jogo no mesmo.

Imagem 14 – Método Especifico

```

//if para falling = false
if(!isJumping) {
    if(!world.isFree(this.getX(), this.getY()-1)) {
        y--;
        jumpframes++;
        if(jumpframes == jumpweight) {
            isJumping = false;
            jump = false;
            jumpframes = 0;
        }
    } else {
        isJumping = false;
        jump = false;
        jumpframes = 0;
    }
}

//Comandos para detectar boneco
for(int i = 0; i < game.entities.size(); i++) {
    Entity e = game.entities.get(i);
    if(e instanceof Boneco) {
        if(!entity.isColliding(this, e)) {
            if(entity.random.nextInt(100) < 5) {
                life--;
            }
            //if para detectar quando o vida player = 0 e mostrar o game over
            if(life <= 0) {
                game.gamestate = "GAME_OVER";
            }
        }
    }
}

//Comandos de colisão dos monstros e aumento da contagem de monstros
for(int i = 0; i < game.entities.size(); i++) {
    Entity e = game.entities.get(i);
    if(e instanceof Plantar || e instanceof Cardboard || e instanceof Glass || e instanceof Paper || e instanceof Seta;) {
        if(!entity.isColliding(this, e)) {
            game.entities.remove(i);
            player.currentCoins++;
            break;
        }
    }
    if(e instanceof Secret) {
        if(!entity.isColliding(this, e)) {
            game.entities.remove(i);
            player.currentSecret++;
            break;
        }
    }
}

//Comandos para inicializar o camera
camera.x = camera.clamp((int)0 - game.WIDTH / 2, 0, world.getWidth() * 10 - game.WIDTH);
camera.y = camera.clamp((int)0 - game.HEIGHT / 2, 0, world.getHeight() * 10 - game.HEIGHT);
}

```

Fonte: IDE Eclipse

Imagem 15 – Método Especifico

```

//Método para o jogo = logica do player
public void tick(){
    depth = 2;
    if(!world.isFree((int)x, (int)(y+gravity)) && isJumping == false) {
        y+=gravity;
        for(int i = 0; i < game.entities.size(); i++) {
            Entity e = game.entities.get(i);
            if(e instanceof Boneco) {
                if(!entity.isColliding(this, e)) {
                    //Comandos de jogo do jogador
                    isJumping = true;
                    jumpweight = 30;

                    //Comandos de remover boneco na inicial
                    ((Enemy) e).vida--;
                    if(((Enemy) e).vida == 0) {
                        //Comandos para matar o inimigo
                        game.entities.remove(i);
                        player.currentEnemies++;
                        break;
                    }
                }
            }
        }
    }
    //if para verificar se o personagem pode ir para a direita
    if(right && world.isFree((int)(x+speed), (int)y)) {
        x+=speed;
        dir = 1;
    }
    //else if para verificar se o personagem pode ir para a esquerda
    else if(left && world.isFree((int)(x-speed), (int)y)) {
        x-=speed;
        dir = -1;
    }
    //if para verificar se o personagem pode
    if(jump) {
        if(!world.isFree(this.getX(), this.getY()+1)) {
            isJumping = true;
        }
        else {
            jump = false;
        }
    }
}

```

Fonte: IDE Eclipse

Método para renderizar os gráficos do jogador.

Imagem 16 – Renderiza Gráficos

```
//Método para renderizar o player
public void render(Graphics g){
    framesAnimation++;
    if(framesAnimation == maxFrames) {
        curSprite++;
        framesAnimation = 0;
        if(curSprite == maxSprite) {
            curSprite = 0;
        }
    }
    //if e else if que atualiza o sprite atual do player
    if(dir == 1) {
        sprite = Entity.PLAYER_SPRITE_RIGHT[curSprite];
    }else if(dir == -1) {
        sprite = Entity.PLAYER_SPRITE_LEFT[curSprite];
    }

    super.render(g);
}
```

Fonte: IDE Eclipse

Na classe entity, terá comandos que obtêm os sprites das entidades por meio da imagem sritesheet, salvando-as num arraylist, facilitando a animação das entidades para serem enviadas as suas referentes classes.

Imagem 17 – Classe Entity

```
// Comandos que sacam o sprite do jogador
public static BufferedImage[] PLAYER_SPRITE_RIGHT = { Game.spritesheet.getSprite(0, 128, 16, 16),
Game.spritesheet.getSprite(15, 128, 16, 16), Game.spritesheet.getSprite(32, 128, 16, 16) };
public static BufferedImage[] PLAYER_SPRITE_LEFT = { Game.spritesheet.getSprite(0, 144, 16, 16),
Game.spritesheet.getSprite(16, 144, 16, 16), Game.spritesheet.getSprite(32, 144, 16, 16) };

// Comandos que sacam o sprite dos inimigos
public static BufferedImage[] ENEMY_RIGHT = { Game.spritesheet.getSprite(32, 0, 16, 16),
Game.spritesheet.getSprite(48, 0, 16, 16), Game.spritesheet.getSprite(64, 0, 16, 16),
Game.spritesheet.getSprite(80, 0, 16, 16) };
public static BufferedImage[] ENEMY_LEFT = { Game.spritesheet.getSprite(32, 0, 16, 16),
Game.spritesheet.getSprite(48, 0, 16, 16), Game.spritesheet.getSprite(64, 0, 16, 16),
Game.spritesheet.getSprite(80, 0, 16, 16) };

// Comandos que sacam os sprites das pacifistas
public static BufferedImage[] PAPER = { Game.spritesheet.getSprite(48, 32, 16, 16),
Game.spritesheet.getSprite(48, 48, 16, 16), Game.spritesheet.getSprite(48, 64, 16, 16),
Game.spritesheet.getSprite(96, 0, 16, 16), Game.spritesheet.getSprite(96, 16, 16, 16),
Game.spritesheet.getSprite(48, 32, 16, 16) };

public static BufferedImage[] GLASS = { Game.spritesheet.getSprite(64, 32, 16, 16),
Game.spritesheet.getSprite(64, 48, 16, 16), Game.spritesheet.getSprite(64, 64, 16, 16),
Game.spritesheet.getSprite(96, 0, 16, 16), Game.spritesheet.getSprite(96, 16, 16, 16),
Game.spritesheet.getSprite(64, 32, 16, 16) };

public static BufferedImage[] CARDBOARD = { Game.spritesheet.getSprite(80, 32, 16, 16),
Game.spritesheet.getSprite(80, 48, 16, 16), Game.spritesheet.getSprite(80, 64, 16, 16),
Game.spritesheet.getSprite(96, 0, 16, 16), Game.spritesheet.getSprite(96, 16, 16, 16),
Game.spritesheet.getSprite(80, 32, 16, 16) };

public static BufferedImage[] PLASTIC = { Game.spritesheet.getSprite(80, 32, 16, 16),
Game.spritesheet.getSprite(96, 48, 16, 16), Game.spritesheet.getSprite(96, 64, 16, 16),
Game.spritesheet.getSprite(96, 0, 16, 16), Game.spritesheet.getSprite(96, 16, 16, 16),
Game.spritesheet.getSprite(80, 32, 16, 16) };

public static BufferedImage[] SECRET = { Game.spritesheet.getSprite(0, 0, 16, 16),
Game.spritesheet.getSprite(0, 0, 16, 16) };

public static BufferedImage[] BATTERY = { Game.spritesheet.getSprite(108, 32, 16, 16),
Game.spritesheet.getSprite(108, 48, 16, 16), Game.spritesheet.getSprite(108, 64, 16, 16),
Game.spritesheet.getSprite(108, 0, 16, 16), Game.spritesheet.getSprite(108, 16, 16, 16),
Game.spritesheet.getSprite(108, 32, 16, 16) };
```

Fonte: IDE Eclipse

Construtor com viáveis de todos os atributos que as entidades necessitam.

Imagem 18 – Construtor com Variáveis

```
// Construtor com variáveis de coordenadas, altura, largura, velocidade e sprite
public Entity(double x, double y, int width, int height, double speed, BufferedImage sprite) {
    this.x = x;
    this.y = y;
    this.speed = speed;
    this.width = width;
    this.height = height;
    this.sprite = sprite;
}
```

Fonte: IDE Eclipse

Método que compara os valores das entidades no arraylist e as organiza.

Imagem 19 – Comparação de Valores

```
// Código usado para comparar duas entidades e organizá-las em um arraylist
public static Comparator<Entity> nodeSorter = new Comparator<Entity>() {

    @Override
    public int compare(Entity n0, Entity n1) {
        if (n1.depth < n0.depth)
            return +1;
        if (n1.depth > n0.depth)
            return -1;
        return 0;
    }

};
```

Fonte: IDE Eclipse

Método que atualiza a câmera, baseado nos valores da tela.

Imagem 20 – Atualização da Câmera

```
// Método que atualiza a câmera
public void updateCamera() {
    Camera.x = Camera.clamp(this.getX() - (Game.WIDTH / 2), 0, World.WIDTH * 16 - Game.WIDTH);
    Camera.y = Camera.clamp(this.getY() - (Game.HEIGHT / 2), 0, World.HEIGHT * 16 - Game.HEIGHT);
}
```

Fonte: IDE Eclipse

Método para verificação da colisão entre as entidades dentro do jogo.

Imagem 21 – Verificar Colisão Entre Entidades

```
// Método para verificar se as entidades estão colidindo
public static boolean isColliding(Entity e1, Entity e2) {
    Rectangle e1Mask = new Rectangle(e1.getX(), e1.getY(), e1.getWidth(), e1.getHeight());
    Rectangle e2Mask = new Rectangle(e2.getX(), e2.getY(), e2.getWidth(), e2.getHeight());

    return e1Mask.intersects(e2Mask);
}
```

Fonte: IDE Eclipse

Na classe enemy, terá um método em que implementa a física por de trás do inimigo, além da própria movimentação do inimigo.

Imagem 22 – Classe Enemy

```
//Método onde é feita a logica do inimigo
public void tick() {
    if(world.isFree((int)x,(int)(y+1))) {
        y++;
    }else {

        //Comandos para o inimigo mover para direita
        if(right) {
            if(world.isFree((int)(x+speed), (int)y)) {
                x+=speed/2;
            }if(world.isFree((int)(x+16),(int)y+1)) {
                right = false;
                left = true;
            }
        }else {
            right = false;
            left = true;
        }
    }

    //Comandos para o inimigo mover para esquerda
    if(left) {
        if(world.isFree((int)(x-speed), (int)y)) {
            x-=speed/2;
        }if(world.isFree((int)(x-16),(int)y+1)) {
            right = true;
            left = false;
        }
    }else {
        right = true;
        left = false;
    }
}
}
```

Fonte: IDE Eclipse

Método em que renderiza todas as sprites do inimigo, como a do mesmo se movendo para esquerda e para direita.

Imagem 23 – Método de Todas Sprites do Inimigo

```
//Método onde renderiza o inimigo
public void render(Graphics g) {

    framesAnimationE++;
    if(framesAnimationE == maxFramesE) {
        curSpriteE++;
        framesAnimationE = 0;
        if(curSpriteE == maxSpriteE) {
            curSpriteE = 0;
        }
    }

    if(right)
        sprite = Entity.ENEMY1_RIGHT[curSpriteE];
    else if(left)
        sprite = Entity.ENEMY1_LEFT[curSpriteE];

    super.render(g);
}
```

Fonte: IDE Eclipse

Nas classes paper, class, cardboard, battery, plastic e secret tem métodos que têm a função de obter as coordenar para posicionar o objeto “paper”, qual é uma bola de papel amassada. Além disso, renderizamos a imagem por meio da obtenção via-imagem spritesheet. Também têm outras cinco classes que possuem o mesmo código, “Glass”, “Cardboard”, “BATTERY”, “Plastic” e “Secret”, quais também são objetivos que interagem com o jogador.

Imagem 24 – Métodos e suas Funções

```
//Metodo com variaveis de coordenadas, altura, largura, velocidade = sprite importando da classe Entity
public Paper(double x, double y, int width, int height, double speed, BufferedImage sprite) {
    super(x, y, width, height, speed, sprite);
}

//Metodo onde renderiza os pixels
public void render(Graphics g) {
    framesAnimationR++;
    if(framesAnimationR == maxFramesR) {
        curSpriteR++;
        framesAnimationR = 0;
        if(curSpriteR == maxSpriteR) {
            curSpriteR = 0;
        }
    }

    sprite = Entity.PAPER[curSpriteR];
    super.render(g);
}
```

Fonte: IDE Eclipse

Este construtor obtém todos os pixels do mapa, organizando-os num array para los valida através da cor hexadecimal para posteriormente substitui-los por um bloco ou entidade, qual tem seus sprites obtidos via-imagem, além de posiciona-los no mapa.



Imagem 25 – Construtor de Todos os Pixels do Mapa

```

public World(String path) {
    try {
        BufferedImage map = ImageIO.read(getClass().getResource(path));
        int[] pixels = new int[map.getWidth() * map.getHeight()]; // array com a quantidade de pixels do mapa
        WIDTH = map.getWidth();
        HEIGHT = map.getHeight();
        tiles = new Tile[map.getWidth() * map.getHeight()];
        map.getRGB(0, 0, map.getWidth(), map.getHeight(), pixels, 0, map.getWidth());

        for (int xx = 0; xx < map.getWidth(); xx++) {
            for (int yy = 0; yy < map.getHeight(); yy++) {
                // Identificação da cor do pixel e qual o tipo de pixel do mapa e qual o nome
                // do sprite que será usado
                int pixelAtual = pixels[xx + (yy * map.getWidth())];

                // Validação dos pixels do chão da cidade
                if ((xx * map.getWidth() + yy * map.getHeight()) == 0x00000000) {
                    tiles[xx + (yy * map.getWidth())] = new FloorTile(xx * 16, yy * 16, Tile.Tile_FLOOR);
                } else if (pixelAtual == 0x00000000) {
                    tiles[xx + (yy * map.getWidth())] = new FloorTile(xx * 16, yy * 16, Tile.Tile_FLOOR);
                }

                // Validação dos pixels das paredes
                if ((xx * map.getWidth() + yy * map.getHeight()) == 0x00000000) {
                    tiles[xx + (yy * map.getWidth())] = new WallTile(xx * 16, yy * 16, Tile.Tile_WALL);
                } else if (pixelAtual == 0x00000000) {
                    tiles[xx + (yy * map.getWidth())] = new WallTile(xx * 16, yy * 16, Tile.Tile_WALL);
                }

                // Validação dos pixels do fundo (céu)
                if ((xx * map.getWidth() + yy * map.getHeight()) == 0x00000000) {
                    tiles[xx + (yy * map.getWidth())] = new WallBlock(xx * 16, yy * 16, Tile.Tile_SKYBLOCK);
                } else if (pixelAtual == 0x00000000) {
                    tiles[xx + (yy * map.getWidth())] = new WallBlock(xx * 16, yy * 16, Tile.Tile_SKYBLOCK);
                }

                // Validação dos pixels da areia
                if ((xx * map.getWidth() + yy * map.getHeight()) == 0x00000000) {
                    tiles[xx + (yy * map.getWidth())] = new FloorSand(xx * 16, yy * 16, Tile.Tile_SAND);
                } else if (pixelAtual == 0x00000000) {
                    tiles[xx + (yy * map.getWidth())] = new FloorSand(xx * 16, yy * 16, Tile.Tile_SAND);
                }
            }
        }
    }
}

```

Fonte: IDE Eclipse

Esse método implementa todo o sistema de colisão de todos os blocos disponíveis no jogo.

Imagem 26 – Implementa o Sistema de Colisão

```

// Sistema de colisão dos tiles
public static boolean isFree(int xnext, int ynext) {
    int x1 = xnext / TILE_SIZE;
    int y1 = ynext / TILE_SIZE;

    int x2 = (xnext + TILE_SIZE - 1) / TILE_SIZE;
    int y2 = (ynext + TILE_SIZE - 1) / TILE_SIZE;

    int x3 = xnext / TILE_SIZE;
    int y3 = (ynext + TILE_SIZE - 1) / TILE_SIZE;

    int x4 = (xnext + TILE_SIZE - 1) / TILE_SIZE;
    int y4 = (ynext + TILE_SIZE - 1) / TILE_SIZE;

    // Validação dos tiles colididos
    return !((tiles[x1 + (y1 * world.WIDTH)] instanceof WallTile) ||
            (tiles[x2 + (y2 * world.WIDTH)] instanceof WallTile) ||
            (tiles[x3 + (y3 * world.WIDTH)] instanceof WallTile) ||
            (tiles[x4 + (y4 * world.WIDTH)] instanceof WallTile) ||

            (tiles[x1 + (y1 * world.WIDTH)] instanceof WallBlock) ||
            (tiles[x2 + (y2 * world.WIDTH)] instanceof WallBlock) ||
            (tiles[x3 + (y3 * world.WIDTH)] instanceof WallBlock) ||
            (tiles[x4 + (y4 * world.WIDTH)] instanceof WallBlock) ||

            (tiles[x1 + (y1 * world.WIDTH)] instanceof FloorSand) ||
            (tiles[x2 + (y2 * world.WIDTH)] instanceof FloorSand) ||
            (tiles[x3 + (y3 * world.WIDTH)] instanceof FloorSand) ||
            (tiles[x4 + (y4 * world.WIDTH)] instanceof FloorSand) ||

            (tiles[x1 + (y1 * world.WIDTH)] instanceof FloorGrass) ||
            (tiles[x2 + (y2 * world.WIDTH)] instanceof FloorGrass) ||
            (tiles[x3 + (y3 * world.WIDTH)] instanceof FloorGrass) ||
            (tiles[x4 + (y4 * world.WIDTH)] instanceof FloorGrass));
}

```

Fonte: IDE Eclipse

Esse método renderiza a visão do jogador (câmera) diante do nível atual.



Imagem 27 – Renderiza a Camera

```
// Metodo de renderização do mapa/level e comandos da camera
public void render(Graphics g) {

    int xstart = Camera.x >> 4;
    int ystart = Camera.y >> 4;

    int xfinal = xstart + (Game.WIDTH >> 4);
    int yfinal = ystart + (Game.HEIGHT >> 4);

    for (int xx = xstart; xx <= xfinal; xx++) {
        for (int yy = ystart; yy <= yfinal; yy++) {
            if (xx < 0 || yy < 0 || xx >= WIDTH || yy >= HEIGHT)
                continue;
            Tile tile = tiles[xx + (yy * WIDTH)];
            tile.render(g);
        }
    }
}
```

Fonte: IDE Eclipse

Na classe “tile” ocorre o carregamento de todos os sprites de blocos, além disso, renderiza-os no jogo.

Imagem 28 – Classe Tile

```
public class Tile {

    //Comando que carrega os sprites dos blocos = sprites estaticos
    public static BufferedImage TILE_FLOOR = Game.spritesheet.getSprite(0,0,16,16);
    public static BufferedImage TILE_WALL = Game.spritesheet.getSprite(16,0,16,16);
    public static BufferedImage TILE_SAND = Game.spritesheet.getSprite(32,0,16,16);
    public static BufferedImage TILE_GRASS = Game.spritesheet.getSprite(48,0,16,16);
    public static BufferedImage TILE_SKYBLOCK = Game.spritesheet.getSprite(0,0,16,16);

    private BufferedImage sprite;
    private int x,y;

    public Tile(int x,int y,BufferedImage sprite){
        this.x = x;
        this.y = y;
        this.sprite = sprite;
    }

    //Metodo de renderização das tiles
    public void render(Graphics g){
        g.drawImage(sprite, x - Camera.x, y - Camera.y, null);
    }
}
```

Fonte: IDE Eclipse

Na classe câmera foi utilizada para limitar o movimento da câmera, qual não permite que ela ultrapasse o tamanho do mapa.

Imagem 29 – Classe Camera

```
public class Camera {

    public static int x = 0;
    public static int y = 0;

    //Metodo que vai ser usado para camera n ultrapassar o tamanho do mapa
    public static int clamp(int Atual,int Min,int Max){
        if(Atual < Min){
            Atual = Min;
        }

        if(Atual > Max) {
            Atual = Max;
        }

        return Atual;
    }
}
```

Fonte: IDE Eclipse

Na classe “FloorTile, FloorSand, FloorGrass, WallBlock e o WallTile” foi utilizado a verificação de colisão das entidades com o blocos, além da “FloorTile” terá outra classe com códigos iguais, como a “FloorSand”, “FloorGrass”, “WallBlock” e o “WallTile”.

Imagem 30 – Floortile

```
//classe para ser usada para verificar colisão
public class FloorTile extends Tile{

    public FloorTile(int x, int y, BufferedImage sprite) {
        super(x, y, sprite);
    }

}
```

Fonte: IDE Eclipse

A classe Background é a que insere o tamanho dos fundos e imagens.

Imagem 31 – Classe Background

```
public class Backgrounds {

    public static BufferedImage BC1 = Game.background1.getSprite(0, 0, 3840, 160);
    public static BufferedImage BC2 = Game.background2.getSprite(0, 0, 3840, 160);
    public static BufferedImage BC3 = Game.background3.getSprite(0, 0, 512, 768);
    public static BufferedImage BCMenu = Game.backgroundMENU.getSprite(0, 0, 720, 480);

}
```

Fonte: IDE Eclipse

Na classe “UI” demonstra e/ou insere na tela todos os valores de vida do protagonista, além dos inimigos derrotados e reciclagens coletadas pelo jogador.

Imagem 32 – Classe “UI”

```
public class UI {

    //Metodo onde mostra a vida e a quantidade de reciclagens coletadas
    public void render(Graphics g) {
        g.setColor(Color.red);
        g.fillRect(10, 10, 150, 30);
        g.setColor(Color.green);
        g.fillRect(10, 10, (int)((Player.life/3) * 150), 30);
        g.setColor(Color.white);
        g.drawRect(10, 10, 150, 30);
        g.setColor(Color.red);
        g.setFont(newFont);
        g.drawString("Reciclagens: "+Player.currentCoins+"/"+Player.maxCoins, (Game.WIDTH*Game.SCALE) - 225, 70);
        g.drawString("Inimigos destruidos: "+Player.currentEnemies+"/"+Player.maxEnemies, (Game.WIDTH*Game.SCALE) - 315, 35);
    }

}
```

Fonte: IDE Eclipse

Este construtor obtém as sprites e as armazena numa variável local, além de ter o método “tick()” que possui controles do jogador.

Imagem 33 – Classe Menu

```
// Construtor do menu - adiciona as opções e as imagens locais
public Menu() {
    player = Game.spritesheet.getSprite(0, 128, 16, 16);
    inimigo = Game.spritesheet.getSprite(48, 0, 16, 16);
}

// Método Tick - Realiza logica por trás do menu
public void tick() {
    // If que controla a tecla de Espaço
    if (DOWN == true) {
        DOWN = false;
        currentOptions++;
        if (currentOptions > maxOptions) {
            currentOptions = 0;
        }
    }

    // If que controla a tecla de seta para cima
    if (UP == true) {
        UP = false;
        currentOptions--;
        if (currentOptions < 0) {
            currentOptions = maxOptions;
        }
    }

    // If que controla a tecla ENTER
    if (OK) {
        OK = false;

        // Muda o gameState para normal
        if (currentOptions == 0) {
            Game.gameState = "NORMAL";
        }

        // Sai do loop
        else if (currentOptions == 1) {
            System.exit(1);
        }
    }
}
}
```

Fonte: IDE Eclipse

Esse método renderiza a “interface” menu para o usuário, além de alterar a posição dos sprites de seleção.

Imagem 34 – Renderiza a Interface

```
public void render(Graphics g) {
    // Renderiza UI do menu
    g.drawImage(Backgrounds.BCMenu, -Camera.x, -Camera.y, 720, 480, null);
    g.setColor(new Color(255, 255, 255));
    g.drawString("Start", 130, 280);
    g.drawString("Exit", 337, 350);
    g.drawImage(player.getScaledInstance(200, 200, 0), 50, 240, null);
    g.drawImage(inimigo.getScaledInstance(200, 200, 0), 510, 250, null);

    // Muda a posição do Sprite da seleção
    if (options[currentOptions] == "Start") {
        g.drawImage(player.getScaledInstance(50, 50, 0), 280, 240, null);
    } else if (options[currentOptions] == "Exit") {
        g.drawImage(player.getScaledInstance(50, 50, 0), 286, 310, null);
    }
}
}
```

Fonte: IDE Eclipse

Essa classe possui a função de adicionar música ao jogo, assim que ele for iniciado.

Imagem 35 – Classe Sound

```

public class Sound {
    // construtor para criar som. Recebe 2 parâmetros: o nome do arquivo e se é loop
    // ou não
    public Sound(final String url, final Boolean loopContinuo) {
        // thread inicializada com o método run
        new Thread(new Runnable() {
            public void run() {
                // try catch, tenta pegar o som da classe chamada, caso contrário mostra erro
                try {
                    Clip clip = AudioSystem.getClip();
                    AudioInputStream inputStream = AudioSystem
                        .getAudioInputStream(getClass().getResourceAsStream(url + ".wav"));
                    clip.open(inputStream);

                    // if caso o loop continue de construtor seja verdadeiro, caso contrário inicia
                    // normalmente
                    if (loopContinuo == true) {
                        clip.loop(Clip.LOOP_CONTINUOUSLY);
                        Thread.sleep(1000);
                    } else {
                        clip.start();
                    }

                } catch (Exception e) {
                    System.err.println(e.getMessage());
                }
            }
        }).start();
    }
}

```

Fonte: IDE Eclipse

## **5 METODOLOGIA**

A escolha do tema se deu devido a importância da reciclagem no meio ambiente. Através disso, ocorre a conscientização das pessoas de que a reciclagem pode ser extremamente útil e saudável para a humanidade e o planeta. E com isso, os dados agregados foram coletados através de mútuas fontes virtuais e pesquisas bibliográficas que possibilitaram consolidar essa pesquisa.

## **6 RESULTADOS**

Nesse trabalho nota-se uma breve introdução ao assunto tido como desenvolvimento sustentável e sua etiologia reciclável. Foi possível adaptar-se a uma forma de que seja entendido diretamente aos internautas, afirmando que a reciclagem pode atingir objetivos de preservação desde que a conscientização humana se torne mais ampla.

## **7 CONSIDERAÇÕES FINAIS**

Este trabalho tem como conclusão que a reciclagem realiza um cargo importante referente ao meio ambiente, já que a mesma defende o planeta terra e gerações futuras a terem uma condição de vida melhor. Além disso, ele garante uma boa conscientização e/ou influência aos internautas a terem boas práticas.

## 8 REFERÊNCIAS

AMBIENTAL. Reciclagem na educação infantil: aprenda como inserir o assunto desde cedo. BRK Ambiental, 2019. <<https://blog.brkambiental.com.br/reciclagem-na-educacao-infantil/>> Acesso em: 04. Mai. 21.

ECOBRAZIL. **Relatório Brundtland – Nosso Futuro em Comum**. Instituto Ecobrazil. <[http://www.ecobrazil.eco.br/site\\_content/30-categoria-conceitos/1003-nosso-futuro-comum-relatorio-brundtland](http://www.ecobrazil.eco.br/site_content/30-categoria-conceitos/1003-nosso-futuro-comum-relatorio-brundtland)> Acesso em: 27. Abr. 21.

ECYCLE. **Entenda os processos por trás da reciclagem de eletrônicos**. Ecycle. <<https://www.ecycle.com.br/1823-reciclagem-de-eletronicos.html>> Acesso em: 04. Mai. 21.

ECYCLE. **O que é reciclagem energética?** Ecycle. <<https://www.ecycle.com.br/3644-reciclagem-energetica.html#:~:text=A%20reciclagem%20energ%C3%A9tica%20%C3%A9%20a,energia%20t%C3%A9rmica%20e%20Fou%20el%C3%A9trica.&text=Entretanto%2C%20o%20material%20descartado%20mais,sua%20utiliza%C3%A7%C3%A3o%20na%20produ%C3%A7%C3%A3o%20energ%C3%A9tica>> Acesso em 04 . Mai. 21.

FRAGMAQ. **Entenda as diferenças entre sustentabilidade fraca e forte**. FRAGMAQ, Jul, 2017. <<https://www.fragmaq.com.br/blog/entenda-as-diferencas-entre-sustentabilidade-fraca-e-forte/#:~:text=A%20sustentabilidade%20forte%2C%20por%20sua,e%20ampliado%20%E2%80%94%20e%20n%C3%A3o%20substitu%C3%ADdo>> Acesso em 27. Abr. 21

INFOWORLD. **Java complete 25 anos: relembre a trajetória, e, afinal, o que podemos esperar para o seu futuro?** ComputerWorld, Mai, 2020. <<https://computerworld.com.br/carreira/java-completa-25-anos-relembre-trajetoria-e-afinal-o-que-podemos-esperar-para-o-seu-futuro/>> Acesso em: 20. Mar. 21.

LUCINIO, Pedro. **Vantagens e desvantagens da linguagem Java**. Programathor. Ago, 2018. <<https://programathor.com.br/blog/vantagens-desvantagens-da-linguagem-java/>> Acesso em: 20. Mar. 21,



MAGALHÃES Lana. **Sustentabilidade.** TODAMATÉRIA.  
<<https://www.todamateria.com.br/sustentabilidade/>> Acesso em 27. Abr. 21.

POLEN. **Lixo e Aquecimento Global: entenda a relação.** Polen, Ago. 2020  
<<https://www.creditologisticareversa.com.br/post/t-lixo-e-aquecimento-global-entenda-a-relacao>> Acesso em: 27. Abr. 21

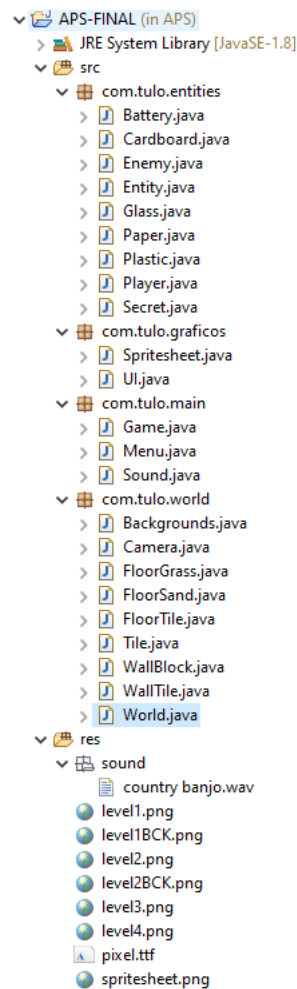
SARDINHA, Vanessa. **10 dicas importantes para preservar o meio ambiente.** MundoEducação. <<https://mundoeducacao.uol.com.br/biologia/10-dicas-importantes-para-preservar-meio-ambiente.htm>> Acesso em: 20. Mar. 21.

SILVA, Gabriel. **História dos jogos mobile: parte 1/3 – dos primeiros jogos ao java.** Blog Gazeus. Jul, 2017. <<https://blog.gazeus.com/uma-breve-hist%C3%B3ria-dos-jogos-mobile-parte-1-3-dos-primeiros-jogos-ao-java-ea03e328a39a?gi=398a4beffd3>>  
Acesso em: 20. Mar. 21.

VGRESÍDUOS. **Reciclagem energética: uma excelente opção para reutilizar resíduos.** VGResíduos, Mai. 2018. <<https://www.vgresiduos.com.br/blog/reciclagem-energetica-uma-excelente-opcao-para-reutilizar-residuos/>> Acesso em 04. Mai. 21

## 9 CÓDIGO FONTE

Imagem 36 – Organização do Código Fonte



Fonte: IDE – Eclipse

Packages    Classes    Código

Com.tulo.entities

Battery

```
package com.tulo.entities;
```

```
import java.awt.Graphics;
```

```
import java.awt.image.BufferedImage;
```

```
public class Battery extends Entity{
```

```
    //Variaveis de animação das reciclagens
```

```
    private int framesAnimationR = 0;
```

```
private int maxFramesR = 10;
```

```
private int curSpriteR = 0;
```

```
private int maxSpriteR = 5;
```

//Metodo com variaveis de coordenadas, altura, largura, velocidade e sprite  
importando da classe Entity

```
public Battery(double x, double y, int width, int height, double speed,  
BufferedImage sprite) {
```

```
    super(x, y, width, height, speed, sprite);
```

```
}
```

//Metodo onde renderiza os papelao

```
public void render(Graphics g) {
```

```
    framesAnimationR++;
```

```
    if(framesAnimationR == maxFramesR) {
```

```
        curSpriteR++;
```

```
        framesAnimationR = 0;
```

```
        if(curSpriteR == maxSpriteR) {
```

```
            curSpriteR = 0;
```

```
        }
```

```
    }
```

```
    sprite = Entity.BATTERY[curSpriteR];
```

```
    super.render(g);
```

```
}
```

```
}
```

```
    CardBoard
```

```
package com.tulo.entities;
```

```
import java.awt.Graphics;
```

```
import java.awt.image.BufferedImage;
```

```

public class Cardboard extends Entity{

    //Variaveis de animação das reciclagens
    private int framesAnimationR = 0;
    private int maxFramesR = 15;
    private int curSpriteR = 0;
    private int maxSpriteR = 3;

    //Metodo com variaveis de coordenadas, altura, largura, velocidade e sprite
    importando da classe Entity
    public Cardboard(double x, double y, int width, int height, double speed,
    BufferedImage sprite) {
        super(x, y, width, height, speed, sprite);
    }

    //Metodo onde renderiza os papelao
    public void render(Graphics g) {

        framesAnimationR++;
        if(framesAnimationR == maxFramesR) {
            curSpriteR++;
            framesAnimationR = 0;
            if(curSpriteR == maxSpriteR) {
                curSpriteR = 0;
            }
        }

        sprite = Entity.CARDBOARD[curSpriteR];

        super.render(g);
    }

}

```

Enemy

```
package com.tulo.entities;
```

```
import java.awt.Graphics;
```

```
import java.awt.image.BufferedImage;
```

```
import com.tulo.world.World;
```

```
public class Enemy extends Entity{
```

```
    private int framesAnimationE = 0;
```

```
    private int maxFramesE = 30;
```

```
    private int curSpriteE = 0;
```

```
    private int maxSpriteE = 4;
```

```
    public boolean right = true,left = false;
```

```
    public int vida = 1;//Variavel da vida do inimigo
```

```
    //Metodo com variaveis de coordenadas, altura, largura e sprite importando da
    classe Entity
```

```
    public Enemy(double x, double y, int width, int height, double speed,
    BufferedImage sprite) {
```

```
        super(x, y, width, height, speed, sprite);
```

```
    }
```

```
    //Metodo onde é feito a logica do inimigo
```

```
    public void tick() {
```

```
        if(World.isFree((int)x,(int)(y+1))) {
```

```
            y+=1;
```

```
        }else {
```

```
            //Comandos para o inimigo mover para direita
```

```
            if(right) {
```

```

        if(World.isFree((int)(x+speed), (int)y)) {
            x+=speed/2;
            if(World.isFree((int)(x+16),(int)y+1)) {
                right = false;
                left = true;
            }
        }else {
            right = false;
            left = true;
        }
    }

    //Comandos para o inimigo mover para esquerda
    if(left) {
        if(World.isFree((int)(x-speed), (int)y)) {
            x-=speed/2;
            if(World.isFree((int)(x-16),(int)y+1)) {
                right = true;
                left = false;
            }
        }else {
            right = true;
            left = false;
        }
    }

}

//Metodo onde renderiza o inimigo
public void render(Graphics g) {

    framesAnimationE++;

```

```

        if(framesAnimationE == maxFramesE) {
            curSpriteE++;
            framesAnimationE = 0;
            if(curSpriteE == maxSpriteE) {
                curSpriteE = 0;
            }
        }

        if(right)
            sprite = Entity.ENEMY1_RIGHT[curSpriteE];
        else if(left)
            sprite = Entity.ENEMY1_LEFT[curSpriteE];

        super.render(g);
    }
}

```

Entity

```
package com.tulo.entities;
```

```
import java.awt.Graphics;
```

```
import java.awt.Rectangle;
```

```
import java.awt.image.BufferedImage;
```

```
import java.util.Comparator;
```

```
import java.util.Random;
```

```
import com.tulo.main.Game;
```

```
import com.tulo.world.Camera;
```

```
import com.tulo.world.World;
```

```
public class Entity {
```

```
    // Comandos que carregam o sprite do jogador
```

```

    public static BufferedImage[] PLAYER_SPRITE_RIGHT = {
Game.spritesheet.getSprite(0, 128, 16, 16),
        Game.spritesheet.getSprite(16, 128, 16, 16),
Game.spritesheet.getSprite(32, 128, 16, 16) };

```

```

    public static BufferedImage[] PLAYER_SPRITE_LEFT = {
Game.spritesheet.getSprite(0, 144, 16, 16),
        Game.spritesheet.getSprite(16, 144, 16, 16),
Game.spritesheet.getSprite(32, 144, 16, 16) };

```

// Comandos que carregam o sprite dos inimigos

```

    public static BufferedImage[] ENEMY1_RIGHT = {
Game.spritesheet.getSprite(32, 0, 16, 16),
        Game.spritesheet.getSprite(48, 0, 16, 16),
Game.spritesheet.getSprite(64, 0, 16, 16),
        Game.spritesheet.getSprite(80, 0, 16, 16) };

```

```

    public static BufferedImage[] ENEMY1_LEFT = {
Game.spritesheet.getSprite(32, 0, 16, 16),
        Game.spritesheet.getSprite(48, 0, 16, 16),
Game.spritesheet.getSprite(64, 0, 16, 16),
        Game.spritesheet.getSprite(80, 0, 16, 16) };

```

// Comandos que carregam os sprites das reciclagens

```

    public static BufferedImage[] PAPER = { Game.spritesheet.getSprite(48, 32, 16,
16),
        Game.spritesheet.getSprite(48, 48, 16, 16),
Game.spritesheet.getSprite(48, 64, 16, 16),
        Game.spritesheet.getSprite(96, 80, 16, 16),
Game.spritesheet.getSprite(96, 96, 16, 16),
        Game.spritesheet.getSprite(48, 32, 16, 16) };

```

```

    public static BufferedImage[] GLASS = { Game.spritesheet.getSprite(64, 32, 16,
16),
        Game.spritesheet.getSprite(64, 48, 16, 16),
Game.spritesheet.getSprite(64, 64, 16, 16),

```



```

        Game.spritesheet.getSprite(96,      80,      16,      16),
Game.spritesheet.getSprite(96, 96, 16, 16),
        Game.spritesheet.getSprite(64, 32, 16, 16) };

```

```

    public static BufferedImage[] CARDBOARD = { Game.spritesheet.getSprite(80,
32, 16, 16),
        Game.spritesheet.getSprite(80,      48,      16,      16),
Game.spritesheet.getSprite(80, 64, 16, 16),
        Game.spritesheet.getSprite(96,      80,      16,      16),
Game.spritesheet.getSprite(96, 96, 16, 16),
        Game.spritesheet.getSprite(80, 32, 16, 16) };

```

```

    public static BufferedImage[] PLASTIC = { Game.spritesheet.getSprite(96, 32,
16, 16),
        Game.spritesheet.getSprite(96,      48,      16,      16),
Game.spritesheet.getSprite(96, 64, 16, 16),
        Game.spritesheet.getSprite(96,      80,      16,      16),
Game.spritesheet.getSprite(96, 96, 16, 16),
        Game.spritesheet.getSprite(96, 32, 16, 16) };

```

```

    public static BufferedImage[] SECRET = { Game.spritesheet.getSprite(0, 0, 16,
16),
        Game.spritesheet.getSprite(0, 0, 16, 16) };

```

```

    public static BufferedImage[] BATTERY = { Game.spritesheet.getSprite(108,
32, 16, 16),
        Game.spritesheet.getSprite(108,      48,      16,      16),
Game.spritesheet.getSprite(108, 64, 16, 16),
        Game.spritesheet.getSprite(108,      80,      16,      16),
Game.spritesheet.getSprite(108, 96, 16, 16),
        Game.spritesheet.getSprite(108, 32, 16, 16) };

```

```

protected double x;

```

```

protected double y;

```

```

protected int width;
protected int height;
protected double speed;

```

```

public int depth;

```

```

public boolean debug = false;

```

```

public BufferedImage sprite;

```

```

public static Random rand = new Random();

```

```

// Construtor com variaveis de coordenadas, altura, largura, velocidade e sprite

```

```

public Entity(double x, double y, int width, int height, double speed,
BufferedImage sprite) {
    this.x = x;
    this.y = y;
    this.speed = speed;
    this.width = width;
    this.height = height;
    this.sprite = sprite;
}

```

```

// Codigos usados para comparar duas entidades e organizalos em um arraylist

```

```

public static Comparator<Entity> nodeSorter = new Comparator<Entity>() {

```

```

    @Override

```

```

    public int compare(Entity n0, Entity n1) {
        if (n1.depth < n0.depth)
            return +1;
        if (n1.depth > n0.depth)
            return -1;
        return 0;
    }
}

```

```
};
```

```
// Metodo que atualiza a camera
```

```
public void updateCamera() {
    Camera.x = Camera.clamp(this.getX() - (Game.WIDTH / 2), 0,
World.WIDTH * 16 - Game.WIDTH);
    Camera.y = Camera.clamp(this.getY() - (Game.HEIGHT / 2), 0,
World.HEIGHT * 16 - Game.HEIGHT);
}
```

```
public void setX(int newX) {
    this.x = newX;
}
```

```
public void setY(int newY) {
    this.y = newY;
}
```

```
public int getX() {
    return (int) this.x;
}
```

```
public int getY() {
    return (int) this.y;
}
```

```
public int getWidth() {
    return this.width;
}
```

```
public int getHeight() {
    return this.height;
}
```

```

public void tick() {
}

// Metodo para verificacao da colisao entre as entidades no jogo
public static boolean isColidding(Entity e1, Entity e2) {
    Rectangle e1Mask = new Rectangle(e1.getX(), e1.getY(), e1.getWidth(),
e1.getHeight());
    Rectangle e2Mask = new Rectangle(e2.getX(), e2.getY(), e2.getWidth(),
e2.getHeight());

    return e1Mask.intersects(e2Mask);
}

// Metodo de renderização da classe Entity
public void render(Graphics g) {
    g.drawImage(sprite, this.getX() - Camera.x, this.getY() - Camera.y, null);
}
}

Glass

package com.tulo.entities;

import java.awt.Graphics;
import java.awt.image.BufferedImage;

public class Glass extends Entity{

    //Variaveis de animação das reciclagens
    private int framesAnimationR = 0;
    private int maxFramesR = 15;
    private int curSpriteR = 0;
    private int maxSpriteR = 3;

```

//Metodo com variaveis de coordenadas, altura, largura, velocidade e sprite  
importando da classe Entity

```
public Glass(double x, double y, int width, int height, double speed,
BufferedImage sprite) {
    super(x, y, width, height, speed, sprite);
}
```

//Metodo onde renderiza os vidros

```
public void render(Graphics g) {

    framesAnimationR++;
    if(framesAnimationR == maxFramesR) {
        curSpriteR++;
        framesAnimationR = 0;
        if(curSpriteR == maxSpriteR) {
            curSpriteR = 0;
        }
    }

    sprite = Entity.GLASS[curSpriteR];

    super.render(g);
}

}
```

Paper

```
package com.tulo.entities;
```

```
import java.awt.Graphics;
```

```
import java.awt.image.BufferedImage;
```

```
public class Paper extends Entity{
```

//Variaveis de animação das reciclagens

```
private int framesAnimationR = 0;
```

```
private int maxFramesR = 15;
```

```
private int curSpriteR = 0;
```

```
private int maxSpriteR = 3;
```

//Metodo com variaveis de coordenadas, altura, largura, velocidade e sprite  
importando da classe Entity

```
public Paper(double x, double y, int width, int height, double speed,  
BufferedImage sprite) {
```

```
    super(x, y, width, height, speed, sprite);
```

```
}
```

//Metodo onde renderiza os papeis

```
public void render(Graphics g) {
```

```
    framesAnimationR++;
```

```
    if(framesAnimationR == maxFramesR) {
```

```
        curSpriteR++;
```

```
        framesAnimationR = 0;
```

```
        if(curSpriteR == maxSpriteR) {
```

```
            curSpriteR = 0;
```

```
        }
```

```
    }
```

```
    sprite = Entity.PAPER[curSpriteR];
```

```
    super.render(g);
```

```
}
```

```
}
```

```
    Plastic
```

```
package com.tulo.entities;
```

```
import java.awt.Graphics;
```

```
import java.awt.image.BufferedImage;
```

```
public class Plastic extends Entity{
```

```
    //Variaveis de animação das reciclagens
```

```
    private int framesAnimationR = 0;
```

```
    private int maxFramesR = 15;
```

```
    private int curSpriteR = 0;
```

```
    private int maxSpriteR = 3;
```

```
    //Metodo com variaveis de coordenadas, altura, largura, velocidade e sprite
    importando da classe Entity
```

```
    public Plastic(double x, double y, int width, int height, double speed,
    BufferedImage sprite) {
```

```
        super(x, y, width, height, speed, sprite);
```

```
    }
```

```
    //Metodo onde renderiza o inimigo
```

```
    public void render(Graphics g) {
```

```
        framesAnimationR++;
```

```
        if(framesAnimationR == maxFramesR) {
```

```
            curSpriteR++;
```

```
            framesAnimationR = 0;
```

```
            if(curSpriteR == maxSpriteR) {
```

```
                curSpriteR = 0;
```

```
            }
```

```
        }
```

```
        sprite = Entity.PLASTIC[curSpriteR];
```

```
        super.render(g);
```

```
    }
```

```
}
```

```
    Player
```

```
package com.tulo.entities;
```

```
import java.awt.Graphics;
```

```
import java.awt.image.BufferedImage;
```

```
import com.tulo.main.Game;
```

```
import com.tulo.world.Camera;
```

```
import com.tulo.world.World;
```

```
public class Player extends Entity{
```

```
    public boolean right,left;
```

```
    public static double life = 3;//Variavel da vida
```

```
    public static int currentCoins = 0;//Variavel da quantidade atual de moedas que  
o jogador pegou
```

```
    public static int maxCoins = 0;//Variavel do maximo de moedas existente
```

```
    public static int currentEnemies = 0;
```

```
    public static int maxEnemies = 0;
```

```
    public static int currentSecret = 0;
```

```
    public int dir = 1;
```

```
    private double gravity = 2;//Varivel da gravidade
```

```
//Variaveis do pulo
```

```
    public boolean jump = false;
```



```

public boolean isJumping = false;
public int jumpHeight = 40;//Altura do pulo
public int jumpFrames = 0;

```

```

private int framesAnimation = 0;
private int maxFrames = 15;//Variavel da velocidade da animação

```

```

private int maxSprite = 3;//Variavel da quantia maxima de sprite usados em
cada direção

```

```

private int curSprite = 0;

```

```

//Metodo com variaveis de coordenadas, altura, largura, velocidade e sprite
importando da classe Entity

```

```

public Player(int x, int y, int width, int height,double speed,BufferedImage
sprite) {
    super(x, y, width, height,speed,sprite);
}

```

```

//Metodo onde é feito a logica do player

```

```

public void tick(){
    depth = 2;
    if(World.isFree((int)x,(int)(y+gravity)) && isJumping == false) {
        y+=gravity;
        for(int i = 0; i < Game.entities.size(); i++) {
            Entity e = Game.entities.get(i);
            if(e instanceof Enemy) {
                if(Entity.isColidding(this, e)) {

```

```

                //Comandos de pulo do jogador

```

```

                isJumping = true;

```

```

                //jumpHeight = 36;

```

```

                //Comandos de causar dano ao inimigo

```

```

                ((Enemy) e).vida--;

```

```

        if(((Enemy) e).vida == 0) {

                                //Comando para matar o inimigo
                                Game.entities.remove(i);
                                Player.currentEnemies++;
                                break;
                                }
        }
    }

    }

}

//If para verificar se é possível andar para a direita
if(right && World.isFree((int)(x+speed), (int)y)) {
    x+=speed;
    dir = 1;
}

//Else if para verificar se é possível andar para a esquerda
else if(left && World.isFree((int)(x-speed), (int)y)) {
    x-=speed;
    dir = -1;
}

//If para verificar se é possível pular
if(jump) {
    if(!World.isFree(this.getX(),this.getY()+1)) {
        isJumping = true;
    }
    else {
        jump = false;
    }
}

//If para realizar o pulo

```

```

if(isJumping) {
    if(World.isFree(this.getX(), this.getY()-2)) {
        y-=2;
        jumpFrames+=2;
        if(jumpFrames == jumpHeight) {
            isJumping = false;
            jump = false;
            jumpFrames = 0;
        }
    } else {
        isJumping = false;
        jump = false;
        jumpFrames = 0;
    }
}

```

//Comandos para detectar dano

```

for(int i = 0; i < Game.entities.size(); i++) {
    Entity e = Game.entities.get(i);
    if(e instanceof Enemy) {
        if(Entity.isColidding(this, e)) {
            if(Entity.rand.nextInt(100) < 5)
                life--;

```

//If para detectar quando a vida chegar 0 e  
mudar a variavel para executar o game over

```

        if(life <= 0) {
            Game.GameState = "GAME_OVER";
        }
    }
}
}

```

```

//Comandos de colisao das moedas e aumento da contagem da mesmas
for(int i = 0; i < Game.entities.size(); i++) {
    Entity e = Game.entities.get(i);
    if(e instanceof Plastic || e instanceof Cardboard || e instanceof
Glass
|| e instanceof Paper || e
instanceof Battery) {

        if(Entity.isColidding(this, e)) {
            Game.entities.remove(i);
            Player.currentCoins++;
            break;
        }
    }
    if(e instanceof Secret) {
        if(Entity.isColidding(this, e)) {
            Game.entities.remove(i);
            Player.currentSecret++;
            break;
        }
    }
}

//Comandos que controlam a camera
Camera.x = Camera.clamp((int)x - Game.WIDTH / 2, 0, World.WIDTH *
16 - Game.WIDTH);
Camera.y = Camera.clamp((int)y - Game.HEIGHT / 2, 0, World.HEIGHT *
16 - Game.HEIGHT);

}

//Metodo para renderizar o player

```

```

public void render(Graphics g){
    framesAnimation++;
    if(framesAnimation == maxFrames) {
        curSprite++;
        framesAnimation = 0;
        if(curSprite == maxSprite) {
            curSprite = 0;
        }
    }
    //If e else if que atualiza o sprite atual do player
    if(dir == 1) {
        sprite = Entity.PLAYER_SPRITE_RIGHT[curSprite];
    }else if(dir == -1) {
        sprite = Entity.PLAYER_SPRITE_LEFT[curSprite];
    }

    super.render(g);
}
}

```

Secret

```
package com.tulo.entities;
```

```
import java.awt.Graphics;
```

```
import java.awt.image.BufferedImage;
```

```
public class Secret extends Entity{
```

```
    //Variaveis de animação das reciclagens
```

```
    private int framesAnimationR = 0;
```

```
    private int maxFramesR = 15;
```

```
    private int curSpriteR = 0;
```

```
    private int maxSpriteR = 2;
```

//Metodo com variaveis de coordenadas, altura, largura, velocidade e sprite  
importando da classe Entity

```
public Secret(double x, double y, int width, int height, double speed,
BufferedImage sprite) {
    super(x, y, width, height, speed, sprite);
}
```

//Metodo onde renderiza o objeto secreto

```
public void render(Graphics g) {

    framesAnimationR++;
    if(framesAnimationR == maxFramesR) {
        curSpriteR++;
        framesAnimationR = 0;
        if(curSpriteR == maxSpriteR) {
            curSpriteR = 0;
        }
    }

    sprite = Entity.SECRET[curSpriteR];

    super.render(g);
}

}
```

com.tulo.graficos

Spritesheet

```
package com.tulo.graficos;
```

```
import java.awt.image.BufferedImage;
```

```
import java.io.IOException;
```

```

import javax.imageio.ImageIO;

public class Spritesheet {

    private BufferedImage spritesheet;

    public Spritesheet(String path)
    {
        try {
            //Variavel que tras a imagem de um diretorio para si
            spritesheet = ImageIO.read(getClass().getResource(path));
        } catch (IOException e) {
            e.printStackTrace();
        }
    }

    //Metodo que obtem a imagem de acordo com as coordenadas e alturas
    public BufferedImage getSprite(int x,int y,int width,int height){
        return spritesheet.getSubimage(x, y, width, height);
    }
}

UI

package com.tulo.graficos;

import java.awt.Color;
import java.awt.Graphics;

import com.tulo.entities.Player;
import com.tulo.main.Game;

public class UI {

```

//Metodo onde mostrar a vida e a quantidade de reciclagens coletadas

```

public void render(Graphics g) {
    g.setColor(Color.red);
    g.fillRect(10, 10,150,30);
    g.setColor(Color.green);
    g.fillRect(10, 10,(int)((Player.life/3) * 150), 30);
    g.setColor(Color.white);
    g.drawRect(10, 10, 150, 30);
    g.setColor(Color.red);
    g.setFont(Game.newFont);
    g.drawString("Reciclagens:
"+Player.currentCoins+"/"+Player.maxCoins,(Game.WIDTH*Game.SCALE) - 225 ,70);
    g.drawString("Inimigos                                destruidos:
"+Player.currentEnemies+"/"+Player.maxEnemies,(Game.WIDTH*Game.SCALE) - 315
,35);
}

}

```

com.tulo.main

Game

```

package com.tulo.main;

import java.awt.Canvas;
import java.awt.Color;
import java.awt.Dimension;
import java.awt.Font;
import java.awt.FontFormatException;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.event.MouseEvent;

```



```

import java.awt.event.MouseListener;
import java.awt.event.MouseMotionListener;
import java.awt.image.BufferStrategy;
import java.awt.image.BufferedImage;
import java.awt.image.DataBufferInt;
import java.io.IOException;
import java.io.InputStream;
import java.util.ArrayList;
import java.util.Collections;
import java.util.List;

```

```

import javax.swing.JFrame;

```

```

import com.tulo.entities.Entity;
import com.tulo.entities.Player;
import com.tulo.graficos.SpriteSheet;
import com.tulo.graficos.UI;
import com.tulo.world.Backgrounds;
import com.tulo.world.Camera;
import com.tulo.world.World;

```

```

public class Game extends Canvas implements Runnable, KeyListener,
MouseListener, MouseMotionListener {

```

```

    private static final long serialVersionUID = 1L;
    public static JFrame frame;
    private Thread thread;
    private boolean isRunning = true;

```

```

    // Variaveis do tamanho da janela
    public static final int WIDTH = 240;
    public static final int HEIGHT = 160;
    public static final int SCALE = 3;

```

```
private BufferedImage image;
```

```
// Variaveis para mudança de nivel
```

```
private int CUR_LEVEL = 1, MAX_LEVEL = 4;
```

```
// Variaveis do game over
```

```
public static String GameState = "MENU";
```

```
private boolean showMessageGameOver = true, restart = false;
```

```
private int framesGameOver = 0;
```

```
// Variaveis do background
```

```
public static Spritesheet background1;
```

```
public static Spritesheet background2;
```

```
public static Spritesheet background3;
```

```
public static Spritesheet backgroundMENU;
```

```
// Variaveis/arrays do menu
```

```
public static Menu menu;
```

```
public static int[] pixels;
```

```
public static int[] lightMap;
```

```
// Variaveis para instanciar classes
```

```
public static World world;
```

```
public static List<Entity> entities;
```

```
public static Spritesheet spritesheet;
```

```
public static Spritesheet bonecodomenu;
```

```
public static Player player;
```

```
public static Sound soundtrack;
```

```
public UI ui;
```

```
// Variaveis para inserir novas fontes
```

```
public InputStream stream =
```

```
ClassLoader.getSystemClassLoader().getResourceAsStream("pixel.ttf");
```

```

        public          InputStream          stream2          =
ClassLoader.getResourceAsStream("pixel.ttf");
        public          InputStream          stream3          =
ClassLoader.getResourceAsStream("pixel.ttf");
        public static Font newFont;
        public static Font newFont2;
        public static Font newFont3;
        // obs: foi feita mais de uma fonte igual mudando apenas o tamanho nos codigos
        // um pouco a baixo

        public Game() {
            addKeyListener(this);
            addMouseListener(this);
            addMouseMotionListener(this);

            // comando para definir o tamanho da janela.
            setPreferredSize(new Dimension(WIDTH * SCALE, HEIGHT * SCALE));

            initFrame();
            image          =          new          BufferedImage(WIDTH,          HEIGHT,
BufferedImage.TYPE_INT_RGB);

            // Comandos para iniciar os objetos.
            spritesheet = new Spritesheet("/spritesheet.png");
            background1 = new Spritesheet("/level1BCK.png");
            background2 = new Spritesheet("/level2BCK.png");
            background3 = new Spritesheet("/level3BCK.png");
            backgroundMENU = new Spritesheet("/backgroundMENU.png");
            entities = new ArrayList<Entity>();
            player = new Player(WIDTH / 2 - 30, HEIGHT / 2, 16, 16, 1.4,
Entity.PLAYER_SPRITE_RIGHT[0]);

            world = new World("/level" + CUR_LEVEL + ".png");
            ui = new UI();

```

```

    lightMap = new int[WIDTH * HEIGHT];
    pixels = ((DataBufferInt) image.getRaster().getDataBuffer()).getData();
    menu = new Menu();// Chama a classe menu

    soundTrack = new Sound("/sound/country banjo", true);

    // Comandos para inserir a nova fonte em diferentes tamanhos
    try {
        newFont      =      Font.createFont(Font.TRUETYPE_FONT,
stream).deriveFont(45f);
        newFont2     =      Font.createFont(Font.TRUETYPE_FONT,
stream2).deriveFont(60f);
        newFont3     =      Font.createFont(Font.TRUETYPE_FONT,
stream3).deriveFont(150f);
    } catch (FontFormatException e) {
        e.printStackTrace();
    } catch (IOException e) {
        e.printStackTrace();
    }

    entities.add(player);

}

// Comandos de configuração da janela
public void initFrame() {
    frame = new JFrame("Plataforma Reciclagem");
    frame.add(this);
    frame.setResizable(false);
    frame.pack();
    frame.setLocationRelativeTo(null);

    // Comando q finaliza o programa sem deixar ele rodando fechado

```

```

        frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
        frame.setVisible(true);
    }

```

// Metodo de start do jogo

```

public synchronized void start() {
    thread = new Thread(this);
    isRunning = true;
    thread.start();
}

```

// Metodo de stop do jogo

```

public synchronized void stop() {
    isRunning = false;
    try {
        thread.join();
    } catch (InterruptedException e) {
        e.printStackTrace();
    }
}

```

// Metodo principal do java

```

public static void main(String args[]) {
    Game game = new Game();
    game.start();
}

```

// Metodo que controla o update e a logica do jogo

```

public void tick() {

    if (GameState == "MENU") {
        menu.tick();
    }
}

```

```

if (GameState == "NORMAL") {
    for (int i = 0; i < entities.size(); i++) {
        Entity e = entities.get(i);
        this.restart = false;
        e.tick();
    }
    // Else if para fazer animação do game over
} else if (GameState == "GAME_OVER") {
    this.framesGameOver++;
    if (framesGameOver == 25) {
        this.framesGameOver = 0;
        if (this.showMessageGameOver) {
            this.showMessageGameOver = false;
        } else {
            this.showMessageGameOver = true;
        }
    }
}

// if para passar de nivel
if (Player.currentCoins == Player.maxCoins) {
    this.CUR_LEVEL++;
    Game.entities.clear();
    entities.add(player);
    Player.currentCoins = 0;
    Player.maxCoins = 0;
    Player.currentSecret = 0;
    Player.currentEnemies = 0;
    Player.maxEnemies = 0;
    Player.life = 3;
    world = new World("/level" + CUR_LEVEL + ".png");

} else if (Player.currentSecret == Player.maxCoins) {
    this.CUR_LEVEL = 3;
    Game.entities.clear();

```

```

        entities.add(player);
        Player.currentCoins = 0;
        Player.maxCoins = 0;
        Player.currentSecret = 0;
        Player.currentEnemies = 0;
        Player.maxEnemies = 0;
        Player.life = 3;
        world = new World("/level3.png");

    } else if (CUR_LEVEL >= MAX_LEVEL) {
        this.CUR_LEVEL = 1;
        Game.entities.clear();
        entities.add(player);
        Player.currentCoins = 0;
        Player.maxCoins = 0;
        Player.currentSecret = 0;
        Player.currentEnemies = 0;
        Player.maxEnemies = 0;
        world = new World("/level1.png");

    }
}

// Metodo de renderização dos graficos
public void render() {

    // Codigos que são usados para retornar o grafico do jogo
    BufferStrategy bs = this.getBufferStrategy();

    if (bs == null) {
        this.createBufferStrategy(3);
        return;
    }
}

```

```

Graphics g = image.getGraphics();
g.setColor(new Color(122, 102, 255));
g.fillRect(0, 0, WIDTH, HEIGHT);

if (CUR_LEVEL == 1)
    g.drawImage(Backgrounds.BC1, -Camera.x, -Camera.y, 3840, 160,
null);

else if (CUR_LEVEL == 2)
    g.drawImage(Backgrounds.BC2, -Camera.x, -Camera.y, 3840, 160,
null);

else if (CUR_LEVEL == 3)
    g.drawImage(Backgrounds.BC3, -Camera.x, -Camera.y, 512, 768,
null);

// Comandos focados na renderização do jogo
world.render(g);
Collections.sort(entities, Entity.nodeSorter);
for (int i = 0; i < entities.size(); i++) {
    Entity e = entities.get(i);
    e.render(g);
}
g.dispose();
g = bs.getDrawGraphics();
g.drawImage(image, 0, 0, WIDTH * SCALE, HEIGHT * SCALE, null);
ui.render(g);

// If para executar menu
if (GameState == "MENU")
    menu.render(g);

// If para executar o game over e aparecer a mensagem de game over e
restarta o
// jogo
if (GameState == "GAME_OVER") {

```



```

Graphics2D g2 = (Graphics2D) g;

// Comandos da escrita do Game Over
g.setFont(Game.newFont3);
g2.setColor(new Color(0, 0, 0, 220));
g2.fillRect(0, 0, WIDTH * SCALE, HEIGHT * SCALE);
g.setColor(Color.red);
g.drawString("GAME OVER", (Game.WIDTH * Game.SCALE) -
550, 210);

g.setFont(Game.newFont2);
g.setColor(Color.red);

// if da animação do ">Pressine Enter..."
if (showMessageGameover)
    g.drawString(">Pressione Enter para reiniciar o jogo<",
(Game.WIDTH * Game.SCALE) - 680, 260);

// Comandos para restarta o level
if (restart == true) {
    GameState = "NORMAL";
    Game.entities.clear();
    entities.add(player);
    world = new World("/level" + CUR_LEVEL + ".png");
    Player.currentCoins = 0;
    Player.life = 3;
    Player.maxCoins = Player.maxCoins / 2;
    Player.currentEnemies = 0;
    Player.maxEnemies = Player.maxEnemies / 2;
    Game.player.updateCamera();
}
}
bs.show();
}

```

```

// Metodo de controle do loop do jogo
public void run() {
    long lastTime = System.nanoTime();
    double amountOfTicks = 60.0; // FPS

    // variavel q vai ser usada para saber o momento certo q o jogo tera q
fazer um

    // update
    double ns = 1000000000 / amountOfTicks;

    double delta = 0;
    int frames = 0;
    double timer = System.currentTimeMillis();
    requestFocus();
    while (isRunning) {
        long now = System.nanoTime();
        delta += (now - lastTime) / ns;
        lastTime = now;

        // IF que controla o FPS
        if (delta >= 1) {
            tick();
            render();
            frames++;
            delta--;
        }

        // If para mostrar no console se realmente esta rodando na
quantidade de FPS

        // informado na variavel amountOfTicks
        if (System.currentTimeMillis() - timer >= 1000) {
            System.out.println("FPS: " + frames);
            frames = 0;
        }
    }
}

```

```

        timer += 1000;
    }

}

stop();
}

// Comandos para controlar o jogador
@Override
public void keyPressed(KeyEvent e) {
    if (e.getKeyCode() == KeyEvent.VK_RIGHT) {
        player.right = true;
    } else if (e.getKeyCode() == KeyEvent.VK_LEFT) {
        player.left = true;
    }

    // Foi usado dois if para o jogador poder pular e ir para uma direção ao
mesmo

    // tempo
    if (e.getKeyCode() == KeyEvent.VK_SPACE) {
        player.jump = true;
    }

    // MENU - up
    if (e.getKeyCode() == KeyEvent.VK_UP) {
        if (GameState == "MENU") {
            menu.UP = true;
        }
    }
    // MENU - down
    } else if (e.getKeyCode() == KeyEvent.VK_DOWN) {
        if (GameState == "MENU") {
            menu.DOWN = true;
        }
    }
}

```

```

    }
    // Enter do reinicio
    if (e.getKeyCode() == KeyEvent.VK_ENTER) {
        restart = true;
        // MENU - ENTER
        if (GameState == "MENU") {
            menu.OK = true;
        }
    }
}

```

// Comandos para detectar quando o usuario parou de aperta as teclas para andar

```

@Override
public void keyReleased(KeyEvent e) {
    if (e.getKeyCode() == KeyEvent.VK_RIGHT) {
        player.right = false;
    } else if (e.getKeyCode() == KeyEvent.VK_LEFT) {
        player.left = false;
    }
}

```

```

@Override
public void keyTyped(KeyEvent e) {
    // TODO Auto-generated method stub
}

```

```

@Override
public void mouseClicked(MouseEvent arg0) {
    // TODO Auto-generated method stub
}

```

```
}
```

```
@Override
```

```
public void mouseEntered(MouseEvent arg0) {
```

```
    // TODO Auto-generated method stub
```

```
}
```

```
@Override
```

```
public void mouseExited(MouseEvent arg0) {
```

```
    // TODO Auto-generated method stub
```

```
}
```

```
@Override
```

```
public void mousePressed(MouseEvent e) {
```

```
}
```

```
@Override
```

```
public void mouseReleased(MouseEvent arg0) {
```

```
    // TODO Auto-generated method stub
```

```
}
```

```
@Override
```

```
public void mouseDragged(MouseEvent arg0) {
```

```
    // TODO Auto-generated method stub
```

```
}
```

```
@Override
```

```
public void mouseMoved(MouseEvent e) {
```

```
}
```

```
}
```

Menu

```
package com.tulo.main;
```

```
import java.awt.Color;
```

```
import java.awt.Graphics;
```

```
import java.awt.image.BufferedImage;
```

```
import com.tulo.world.Backgrounds;
```

```
import com.tulo.world.Camera;
```

```
public class Menu {
```

```
    // inicializa variaveis e atributos
```

```
    public String[] options = { "Start", "Exit" };
```

```
    public int currentOptions = 0;
```

```
    public int maxOptions = options.length - 1;
```

```
    public boolean UP, DOWN, OK;
```

```
    public BufferedImage player;
```

```
    public BufferedImage inimigo;
```

```
    public static Sound soundTrack;
```

```
    public boolean pause = false;
```

```
    // Construtor do menu - adiciona as Sprites aos atributos locais
```

```
    public Menu() {
```

```
        player = Game.spritesheet.getSprite(0, 128, 16, 16);
```

```
        inimigo = Game.spritesheet.getSprite(48, 0, 16, 16);
```

```
    }
```

```
    // Metodo Tick - Realiza logica por tras do menu
```

```
    public void tick() {
```

```

// If que controla a tecla de Descer
if (DOWN == true) {
    DOWN = false;
    currentOptions++;
    if (currentOptions > maxOptions) {
        currentOptions = 0;
    }
}

// If que controla a tecla de Subir
if (UP == true) {
    UP = false;
    currentOptions--;
    if (currentOptions < 0) {
        currentOptions = maxOptions;
    }
}

// If que controla a tecla ENTER
if (OK) {
    OK = false;

    // muda o gameState pra normal
    if (currentOptions == 0) {
        Game.GameState = "NORMAL";

        //sai do jogo
    } else if (currentOptions == 1) {
        System.exit(1);
    }
}
}

public void render(Graphics g) {

```

```

// renderiza UI do menu

g.drawImage(Backgrounds.BCmenu, -Camera.x, -Camera.y, 720, 480,
null);

g.setColor(new Color(255, 255, 255));
g.drawString("Start", 330, 280);
g.drawString("Exit", 337, 350);
g.drawImage(player.getScaledInstance(200, 200, 0), 50, 240, null);
g.drawImage(inimigo.getScaledInstance(200, 200, 0), 510, 250, null);

// muda a posicao do Sprite da selecao
if (options[currentOptions] == "Start") {
    g.drawImage(player.getScaledInstance(50, 50, 0), 280, 240, null);
} else if (options[currentOptions] == "Exit") {
    g.drawImage(player.getScaledInstance(50, 50, 0), 286, 310, null);
}
}
}

Sound
package com.tulo.main;

import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.Clip;

public class Sound {

    // Construtor para rodar som. Recebe 2 parametros o nome do arquivo e se é
loop
// ou não
public Sound(final String url, final Boolean loopContinuo) {
    // thread inicializada com o metodo run
    new Thread(new Runnable() {

```



```

    public void run() {
        // try catch, tenta pegar o som da classe chamada, caso
        contrario mostra erro.

        try {
            Clip clip = AudioSystem.getClip();
            AudioInputStream inputStream = AudioSystem

            .getAudioInputStream(Game.class.getResourceAsStream(url + ".wav"));
            clip.open(inputStream);

            // if caso o loop continuo do construtor seja
            verdadeiro, caso contrario inicia

            // normalmente
            if (loopContinuo == true) {
                clip.loop(Clip.LOOP_CONTINUOUSLY);
                Thread.sleep(10000);
            } else {
                clip.start();
            }

        } catch (Exception e) {
            System.err.println(e.getMessage());
        }

    }

    }).start();
}
}

```

com.tulo.world

Backgrounds

**package** com.tulo.world;

```
import java.awt.image.BufferedImage;
```

```
import com.tulo.main.Game;
```

```
//Classe que insere o tamanho dos fundos e imagens.
```

```
public class Backgrounds {
```

```
    public static BufferedImage BC1 = Game.background1.getSprite(0, 0, 3840,  
160);
```

```
    public static BufferedImage BC2 = Game.background2.getSprite(0, 0, 3840,  
160);
```

```
    public static BufferedImage BC3 = Game.background3.getSprite(0, 0, 512,  
768);
```

```
    public static BufferedImage BCmenu = Game.backgroundMENU.getSprite(0, 0,  
720, 480);  
}
```

Camera

```
package com.tulo.world;
```

```
public class Camera {
```

```
    public static int x = 0;
```

```
    public static int y = 0;
```

```
//Metodo que vai ser usado para camera n ultrapassar o tamanho do mapa
```

```
public static int clamp(int Atual,int Min,int Max){
```

```
    if(Atual < Min){  
        Atual = Min;  
    }
```

```
    if(Atual > Max) {  
        Atual = Max;  
    }
```

```

        return Atual;
    }

}

FloorGrass
package com.tulo.world;

import java.awt.image.BufferedImage;

//Classe para ser usada para verificar colições
public class FloorGrass extends Tile{

    public FloorGrass(int x, int y, BufferedImage sprite) {
        super(x, y, sprite);
    }
}

```

```

FloorSand
package com.tulo.world;

import java.awt.image.BufferedImage;

//Classe para ser usada para verificar colições
public class FloorSand extends Tile{

    public FloorSand(int x, int y, BufferedImage sprite) {
        super(x, y, sprite);
    }
}

```

```

FloorTile
package com.tulo.world;

```

```
import java.awt.image.BufferedImage;
```

```
//Classe para ser usada para verificar colições
```

```
public class FloorTile extends Tile{
```

```
    public FloorTile(int x, int y, BufferedImage sprite) {
        super(x, y, sprite);
    }
}
```

```
Tile
```

```
package com.tulo.world;
```

```
import java.awt.Graphics;
```

```
import java.awt.image.BufferedImage;
```

```
import com.tulo.main.Game;
```

```
public class Tile {
```

```
    //Comando que carregam os sprites dos blocos e objetos estaticos
```

```
    public          static          BufferedImage      TILE_FLOOR          =
Game.spritesheet.getSprite(0,0,16,16);
```

```
    public          static          BufferedImage      TILE_WALL          =
Game.spritesheet.getSprite(16,0,16,16);
```

```
    public          static          BufferedImage      TILE_SAND          =
Game.spritesheet.getSprite(32,16,16,16);
```

```
    public          static          BufferedImage      TILE_GRASS          =
Game.spritesheet.getSprite(48,16,16,16);
```

```
    public          static          BufferedImage      TILE_SKYBLOCK       =
Game.spritesheet.getSprite(0,0,16,16);
```

```
    private BufferedImage sprite;
```

```
private int x,y;
```

```
public Tile(int x,int y,BufferedImage sprite){
```

```
    this.x = x;
```

```
    this.y = y;
```

```
    this.sprite = sprite;
```

```
}
```

```
//Metodo de renderização dos tiles
```

```
public void render(Graphics g){
```

```
    g.drawImage(sprite, x - Camera.x, y - Camera.y, null);
```

```
}
```

```
}
```

```
WallBlock
```

```
package com.tulo.world;
```

```
import java.awt.image.BufferedImage;
```

```
//Classe para ser usada para verificar colisões
```

```
public class WallBlock extends Tile{
```

```
    public WallBlock(int x, int y, BufferedImage sprite) {
```

```
        super(x, y, sprite);
```

```
    }
```

```
}
```

```
WallTile
```

```
package com.tulo.world;
```

```
import java.awt.image.BufferedImage;
```

```
//Classe para ser usada para verificar colisões
```

```

public class WallTile extends Tile{

    public WallTile(int x, int y, BufferedImage sprite) {
        super(x, y, sprite);
    }

}

World

package com.tulo.world;

import java.awt.Graphics;
import java.awt.image.BufferedImage;
import java.io.IOException;

import javax.imageio.ImageIO;

import com.tulo.entities.Battery;
import com.tulo.entities.Cardboard;
import com.tulo.entities.Enemy;
import com.tulo.entities.Entity;
import com.tulo.entities.Glass;
import com.tulo.entities.Paper;
import com.tulo.entities.Plastic;
import com.tulo.entities.Player;
import com.tulo.entities.Secret;
import com.tulo.main.Game;

public class World {

    public static Tile[] tiles; // array dos tiles
    public static int WIDTH, HEIGHT; // Variaveis de altura e largura
    public static final int TILE_SIZE = 16; // Variavel do tamanho tile

```

```

public BufferedImage sprite;

public World(String path) {
    try {

        BufferedImage map =
        ImageIO.read(getClass().getResource(path));

        int[] pixels = new int[map.getWidth() * map.getHeight()]; // Array
        com a quantidade de pixels do mapa
        WIDTH = map.getWidth();
        HEIGHT = map.getHeight();
        tiles = new Tile[map.getWidth() * map.getHeight()];
        map.getRGB(0, 0, map.getWidth(), map.getHeight(), pixels, 0,
        map.getWidth());

        for (int xx = 0; xx < map.getWidth(); xx++) {

            for (int yy = 0; yy < map.getHeight(); yy++) {

                // Variavel do pixel atual onde é usado para ver a cor
                do pixel do mapa e colocar

                // o sprite certo naquele pixel
                int pixelAtual = pixels[xx + (yy * map.getWidth())];

                // Validação dos pixels do Chao da cidade
                tiles[xx + (yy * WIDTH)] = new FloorTile(xx * 16, yy *
                16, Tile.TILE_FLOOR);

                if (pixelAtual == 0xFF000000) {
                    tiles[xx + (yy * WIDTH)] = new FloorTile(xx *
                    16, yy * 16, Tile.TILE_FLOOR);

                    // Validação dos pixels das paredes
                } else if (pixelAtual == 0xFFFFFFFF) {

```

```

        tiles[xx + (yy * WIDTH)] = new WallTile(xx * 16,
yy * 16, Tile.TILE_WALL);

        if (yy - 1 >= 0 && pixels[xx + ((yy - 1) *
map.getWidth())] == 0xFFffffff) {
            tiles[xx + (yy * WIDTH)] = new
WallTile(xx * 16, yy * 16,

Game.spritesheet.getSprite(16, 0, 16, 16));
        }

        // Validação dos pixels do fundo invisível q n pd
passar
    } else if (pixelAtual == 0xFF808080) {
        tiles[xx + (yy * WIDTH)] = new WallBlock(xx *
16, yy * 16, Tile.TILE_SKYBLOCK);

        if (yy - 1 >= 0 && pixels[xx + ((yy - 1) *
map.getWidth())] == 0xFF808080) {
            tiles[xx + (yy * WIDTH)] = new
WallBlock(xx * 16, yy * 16,

Game.spritesheet.getSprite(0, 0, 16, 16));
        }

        // Validação dos pixels da Areia
    } else if (pixelAtual == 0xFF7ff68) {
        tiles[xx + (yy * WIDTH)] = new FloorSand(xx *
16, yy * 16, Tile.TILE_SAND);

        if (yy - 1 >= 0 && pixels[xx + ((yy - 1) *
map.getWidth())] == 0xFF7ff68) {
            tiles[xx + (yy * WIDTH)] = new
FloorSand(xx * 16, yy * 16,

```



```

Game.spritesheet.getSprite(32, 16, 16, 16));
    }

    // Validação dos pixels da Grama
} else if (pixelAtual == 0xFF00ff04) {
    tiles[xx + (yy * WIDTH)] = new FloorGrass(xx *
16, yy * 16, Tile.TILE_GRASS);

    if (yy - 1 >= 0 && pixels[xx + ((yy - 1) *
map.getWidth())] == 0xFF00ff04) {
        tiles[xx + (yy * WIDTH)] = new
FloorGrass(xx * 16, yy * 16,

Game.spritesheet.getSprite(48, 16, 16, 16));
    }

    // Validação dos pixels de spawn do Jogador
} else if (pixelAtual == 0xFF0026FF) {
    Game.player.setX(xx * 16);
    Game.player.setY(yy * 16);

    // Validação dos pixels de spawn dos Inimigos
} else if (pixelAtual == 0xFFFF0000) {
    Enemy enemy = new Enemy(xx * 16, yy * 16,
16, 16, 1, Entity.ENEMY1_RIGHT[1]);
    Game.entities.add(enemy);
    Player.maxEnemies++;

    // Validação dos pixels de spawn das
Reciclagem
} else if (pixelAtual == 0xFFFFD800) {
    Paper paper = new Paper(xx * 16, yy * 16, 16,
16, 1, Entity.PAPER[1]);

```

```

        Game.entities.add(paper);
        Player.maxCoins++;
    } else if (pixelAtual == 0xFF00ff21) {
        Glass glass = new Glass(xx * 16, yy * 16, 16,
16, 1, Entity.GLASS[1]);

        Game.entities.add(glass);
        Player.maxCoins++;
    } else if (pixelAtual == 0xFF00f6ff) {
        Cardboard cardboard = new Cardboard(xx *
16, yy * 16, 16, 16, 1, Entity.CARDBOARD[1]);

        Game.entities.add(cardboard);
        Player.maxCoins++;
    } else if (pixelAtual == 0xFFff00e9) {
        Plastic plastic = new Plastic(xx * 16, yy * 16,
16, 16, 1, Entity.PLASTIC[1]);

        Game.entities.add(plastic);
        Player.maxCoins++;
    } else if (pixelAtual == 0xFF5d00ff) {
        Battery battery = new Battery(xx * 16, yy * 16,
16, 16, 1, Entity.BATTERY[1]);

        Game.entities.add(battery);
        Player.maxCoins++;

        // Validação do pixels do secreto
    } else if (pixelAtual == 0xFFff927f) {
        Secret secret = new Secret(xx * 16, yy * 16, 16,
16, 1, Entity.SECRET[1]);

        Game.entities.add(secret);
    }
}
}
} catch (IOException e) {
    e.printStackTrace();
}

```

```
}
```

```
// Sistema de colisão dos tiles
```

```
public static boolean isFree(int xnext, int ynext) {
```

```
    int x1 = xnext / TILE_SIZE;
```

```
    int y1 = ynext / TILE_SIZE;
```

```
    int x2 = (xnext + TILE_SIZE - 1) / TILE_SIZE;
```

```
    int y2 = ynext / TILE_SIZE;
```

```
    int x3 = xnext / TILE_SIZE;
```

```
    int y3 = (ynext + TILE_SIZE - 1) / TILE_SIZE;
```

```
    int x4 = (xnext + TILE_SIZE - 1) / TILE_SIZE;
```

```
    int y4 = (ynext + TILE_SIZE - 1) / TILE_SIZE;
```

```
// Validação dos tiles colidindo
```

```
return !((tiles[x1 + (y1 * World.WIDTH)] instanceof WallTile)
```

```
    || (tiles[x2 + (y2 * World.WIDTH)] instanceof WallTile)
```

```
    || (tiles[x3 + (y3 * World.WIDTH)] instanceof WallTile)
```

```
    || (tiles[x4 + (y4 * World.WIDTH)] instanceof WallTile) ||
```

```
    (tiles[x1 + (y1 * World.WIDTH)] instanceof WallBlock)
```

```
    || (tiles[x2 + (y2 * World.WIDTH)] instanceof WallBlock)
```

```
    || (tiles[x3 + (y3 * World.WIDTH)] instanceof WallBlock)
```

```
    || (tiles[x4 + (y4 * World.WIDTH)] instanceof WallBlock) ||
```

```
    (tiles[x1 + (y1 * World.WIDTH)] instanceof FloorSand)
```

```
    || (tiles[x2 + (y2 * World.WIDTH)] instanceof FloorSand)
```

```
    || (tiles[x3 + (y3 * World.WIDTH)] instanceof FloorSand)
```

```
    || (tiles[x4 + (y4 * World.WIDTH)] instanceof FloorSand) ||
```

```
    (tiles[x1 + (y1 * World.WIDTH)] instanceof FloorGrass)
```

```

        || (tiles[x2 + (y2 * World.WIDTH)] instanceof FloorGrass)
        || (tiles[x3 + (y3 * World.WIDTH)] instanceof FloorGrass)
        || (tiles[x4 + (y4 * World.WIDTH)] instanceof FloorGrass));
    }

```

// Este método renderiza a visão do jogador (câmera) diante o nível atual.

```


public void render(Graphics g) {

    int xstart = Camera.x >> 4;
    int ystart = Camera.y >> 4;

    int xfinal = xstart + (Game.WIDTH >> 4);
    int yfinal = ystart + (Game.HEIGHT >> 4);

    for (int xx = xstart; xx <= xfinal; xx++) {
        for (int yy = ystart; yy <= yfinal; yy++) {
            if (xx < 0 || yy < 0 || xx >= WIDTH || yy >= HEIGHT)
                continue;
            Tile tile = tiles[xx + (yy * WIDTH)];
            tile.render(g);
        }
    }
}

```











FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: Gilberto de Carvalho Almeida Júnior
TURMA: CC3P17
RA: F202453

CURSO: Ciências da Computação
CAMPUS: Sorocaba
SEMESTRE: 3º Semestre
TURNO: Noturno

CÓDIGO DA ATIVIDADE: 76B9
SEMESTRE: 3º Semestre
ANO GRADE: 2021

DATA DA ATIVIDADE	DESCRIÇÃO DA ATIVIDADE	TOTAL DE HORAS	ASSINATURA DO ALUNO	HORAS ATRIBUÍDAS (1)	ASSINATURA DO PROFESSOR
11/03 - 23/05	Reuniões (Brainstorm, decisões, entre outras)	5h			
11/03 - 11/05	Divisão das Funções	2h			
15/03 - 20/05	Pesquisa Referencial Teórica	10h			
20/03 - 20/05	Estudo do Jogo em Java	20h			
27/03 - 22/05	Criação do Código Fonte	30h			
20/04 - 23/05	Escrita e Formatação da ABNT	20h			
10/05 - 20/05	Pixel Art	10h			
18/05 - 18/05	PowerPoint	1h			

[1] Horas atribuídas de acordo com o regulamento das Atividades Práticas Supervisionadas do curso.

TOTAL DE HORAS ATRIBUÍDAS: 98 horas

AVALIAÇÃO:

Aprovado ou Reprovado

NOTA:

DATA: / /

CARIMBO E ASSINATURA DO COORDENADOR DO CURSO



## FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: Pedro Henrique Terreiro de Arruda TURMA: CC2P17 RA: N686650

CURSO: Ciência da Computação CAMPUS: Sorocaba (17) SEMESTRE: 2º semestre TURNO: Noturno

CÓDIGO DA ATIVIDADE: 7689 SEMESTRE: 2º ANO GRADE: 2021

DATA DA ATIVIDADE	DESCRIÇÃO DA ATIVIDADE	TOTAL DE HORAS	ASSINATURA DO ALUNO	HORAS ATRIBUÍDAS (1)	ASSINATURA DO PROFESSOR
11/03 - 23/03	Reunião	5hrs			
11/03 - 11/03	Divisão das Funções	2hrs			
20/03 - 20/03	Estudo do Jogo em Java	20hrs			
27/03 - 22/03	Criação do Código Fonte	30hrs			
20/04 - 23/04	Escrita e Formatação da ABNT	20hrs			
10/05 - 20/05	Pixel Art	10hrs			
18/05 - 18/05	PowerPoint	1hr			
15/03 - 20/05	Pesquisa Referencial Teórico	10hrs			

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

TOTAL DE HORAS ATRIBUÍDAS: 88

AVALIAÇÃO: \_\_\_\_\_

Aprovado ou Reprovado

NOTA: \_\_\_\_\_

DATA: \_\_\_\_/\_\_\_\_/\_\_\_\_

&lt; (Portugal)



## FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: Carlos Eduardo Marquetti Correa Da Silva TURMA: CC2P17 RA: F33IEF7

CURSO: Ciência da Computação CAMPUS: Sorocaba (17) SEMESTRE: 2º Semestre

TURNO: Noturno

CÓDIGO DA ATIVIDADE: 7689 SEMESTRE: 2º Semestre ANO GRADE: 2021

DATA DA ATIVIDADE	DESCRIÇÃO DA ATIVIDADE	TOTAL DE HORAS	ASSINATURA DO ALUNO	HORAS ATRIBUÍDAS (1)	ASSINATURA DO PROFESSOR
11/03 - 23/03	Reunião	5h			
11/03 - 11/03	Divisão das Funções	2h			
20/03 - 20/03	Estudo do Jogo em Java	20h			
27/03 - 22/03	Criação do Código Fonte	30h			
20/04 - 23/04	Escrita e Formatação da ABNT	20h			
10/05 - 20/05	Pixel Art	10h			
18/05 - 20/05	PowerPoint	1h			
15/03 - 20/05	Pesquisa Referencial Teórico	10h			

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

TOTAL DE HORAS ATRIBUÍDAS: 98h

AVALIAÇÃO: \_\_\_\_\_

Aprovado ou Reprovado

NOTA: \_\_\_\_\_

DATA: \_\_\_\_/\_\_\_\_/\_\_\_\_

CARIMBO E ASSINATURA DO COORDENADOR DO CURSO



## FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: Thiago de Paula Souza TURMA: CC2P17 RA: F33JIC8  
 CURSO: Ciência da Computação CAMPUS: Sorocaba (17) SEMESTRE: 2º Semestre TURNO: Noturno  
 CÓDIGO DA ATIVIDADE: 76B9 SEMESTRE: 2º Semestre ANO GRADE: 2021

DATA DA ATIVIDADE	DESCRIÇÃO DA ATIVIDADE	TOTAL DE HORAS	ASSINATURA DO ALUNO	HORAS ATRIBUÍDAS (1)	ASSINATURA DO PROFESSOR
11/03 - 23/05	Reunião	5h	<i>Thiago P. Souza</i>		
11/03 - 11/03	Divisão das Funções	2h	<i>Thiago P. Souza</i>		
20/03 - 20/05	Estudo do Jogo em Java	20h	<i>Thiago P. Souza</i>		
27/03 - 22/05	Criação do Código Fonte	30h	<i>Thiago P. Souza</i>		
20/04 - 23/05	Escrita e Formatação da ABNT	20h	<i>Thiago P. Souza</i>		
10/05 - 20/05	Pixel Art	10h	<i>Thiago P. Souza</i>		
18/05 - 20/05	PowerPoint	1h	<i>Thiago P. Souza</i>		
15/03 - 20/05	Pesquisa Referencial Teórico	10h	<i>Thiago P. Souza</i>		

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

TOTAL DE HORAS ATRIBUÍDAS: 98hrs

AValiação: \_\_\_\_\_ Aprovado ou Reprovado

NOTA: \_\_\_\_\_

DATA: \_\_\_\_/\_\_\_\_/\_\_\_\_

CARIMBO E ASSINATURA DO COORDENADOR DO CURSO



## FICHA DAS ATIVIDADES PRÁTICAS SUPERVISIONADAS - APS

NOME: Yan Gabriel Silva Queiroz TURMA: CC3Q17 RA: N625414  
 CURSO: Ciência da Computação CAMPUS: Sorocaba SEMESTRE: 3º Semestre TURNO: Noturno  
 CÓDIGO DA ATIVIDADE: 76B9 SEMESTRE: 3º Semestre ANO GRADE: 2021

DATA DA ATIVIDADE	DESCRIÇÃO DA ATIVIDADE	TOTAL DE HORAS	ASSINATURA DO ALUNO	HORAS ATRIBUÍDAS (1)	ASSINATURA DO PROFESSOR
11/03 - 23/05	Reunião	5hrs	<i>Yan Gabriel Silva Queiroz</i>		
11/03 - 11/03	Divisão das Funções	2hrs	<i>Yan Gabriel Silva Queiroz</i>		
20/03 - 20/05	Estudo do Jogo em Java	20hrs	<i>Yan Gabriel Silva Queiroz</i>		
27/03 - 22/05	Criação do Código Fonte	30hrs	<i>Yan Gabriel Silva Queiroz</i>		
20/04 - 23/05	Escrita e Formatação da ABNT	20hrs	<i>Yan Gabriel Silva Queiroz</i>		
10/05 - 20/05	Pixel Art	10hrs	<i>Yan Gabriel Silva Queiroz</i>		
18/05 - 18/05	PowerPoint	1hr	<i>Yan Gabriel Silva Queiroz</i>		
25/03 - 20/05	Pesquisa Referencial Teórico	10hrs	<i>Yan Gabriel Silva Queiroz</i>		

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

TOTAL DE HORAS ATRIBUÍDAS: 98hrs

AValiação: \_\_\_\_\_ Aprovado ou Reprovado

NOTA: \_\_\_\_\_

DATA: \_\_\_\_/\_\_\_\_/\_\_\_\_

CARIMBO E ASSINATURA DO COORDENADOR DO CURSO