Uma imagem com sentado, preto, computador

Descrição gerada automaticamente

**Faculdade de Ciências Exatas e de Engenharia**

2019/2020

**Programação Orientada por Objetos**

**Alterações Climáticas**

**Trabalho realizado por:**

Diego Andrés da Silva Briceño (nº 2043818)

Sílvia da Silva Fernandes (nº 2043118)

Rúben José Gouveia Rodrigues (nº 2046018)

Funchal, 17 de março de 2020

**Índice**

[1. Introdução 5](#_Toc35418909)

[2. Ideia Principal do Jogo 5](#_Toc35418910)

[3. Procedimento e implementação do código 5](#_Toc35418911)

[3.1. Menus e Escolhas 5](#_Toc35418912)

[3.1.1. Classe *World* 5](#_Toc35418913)

[3.1.1.1 Subclasse “MenuInicial” 5](#_Toc35418914)

[3.1.1.2. Subclasse “Opções” 6](#_Toc35418915)

[3.1.1.3. Subclasse “EscolhaNomes” 6](#_Toc35418916)

[3.1.1.4. Subclasse “EscolherCor” 6](#_Toc35418917)

[3.1.1.5. Subclasse “HowToPlay” 6](#_Toc35418918)

[3.1.2. Classe Actor e Classe Menus 7](#_Toc35418919)

[3.1.2.1. Subclasse “Play” 7](#_Toc35418920)

[3.1.2.2. Subclasse “Options” 7](#_Toc35418921)

[3.1.2.3. Subclasse “Exit” 7](#_Toc35418922)

[3.1.2.4. Subclasse “Controlos” 7](#_Toc35418923)

[3.1.2.5. Subclasse “Texto” 7](#_Toc35418924)

[3.1.2.6. Subclasse “Back” 8](#_Toc35418925)

[3.2. Jogo 1 8](#_Toc35418926)

[3.2.1. Classe *World* 8](#_Toc35418927)

[3.2.1.1 Subclasse “Jogo1” 8](#_Toc35418928)

[3.2.2 Classe *Actor* >> Subclasse “Objetos” >> Subclasse “Jogos” 10](#_Toc35418929)

[3.2.2.1 Subclasse “ObjetosCaem” 10](#_Toc35418930)

[3.2.2.1.1 Subclasses Granizo, Gota e Neve 10](#_Toc35418931)

[3.2.2.1.2 Subclasse “Vida” 10](#_Toc35418932)

[3.2.2.2 Subclasse “Target” 10](#_Toc35418933)

[3.2.2.3 Subclasse “Relâmpago” 11](#_Toc35418934)

[3.2.2.4 Subclasse “Nuvem” 11](#_Toc35418935)

[3.2.2.5 Subclasse “Máquina” 11](#_Toc35418936)

[3.2.2.5.1 Subclasse “VidaMáquina” 12](#_Toc35418937)

[3.2.2.6 Subclasse “Chão” 13](#_Toc35418938)

[3.2.2.7 Subclasse “Gás” 13](#_Toc35418939)

[3.2.3 Classe *Actor* 13](#_Toc35418940)

[3.2.3.1 Subclasse “Players” 13](#_Toc35418941)

[3.2.3.1.1 Subclasse “Player1” e “Player2” 14](#_Toc35418942)

[3.2.3.1.1.1 Subclasses “Vida\_player1” e “Vida\_player2” 15](#_Toc35418943)

[3.2.3.1.2 Subclasse “Bala” 16](#_Toc35418944)

[3.3. Jogo 2 16](#_Toc35418945)

[3.3.1. Classe *World* 17](#_Toc35418946)

[3.3.1.1. Subclasse “Jogo2” 17](#_Toc35418947)

[3.3.2. Classe *Actor* 17](#_Toc35418948)

[3.3.2.1. Subclasses “Nave1” e “Nave2” 17](#_Toc35418949)

[3.3.2.2. Subclasse “Gas” 18](#_Toc35418950)

[3.3.2.3. Subclasse “CamadaOzono” 18](#_Toc35418951)

[3.3.2.4. Subclasse “VidaCamadaOzono” 18](#_Toc35418952)

[3.3.2.5. Subclasse “Missil” 18](#_Toc35418953)

[3.4. Jogo 3 19](#_Toc35418954)

[3.4.1. Classe *World* 19](#_Toc35418955)

[3.4.1.1. Subclasse “Jogo3” 19](#_Toc35418956)

[3.4.2. Classe *Actor* 20](#_Toc35418957)

[3.4.2.1. Subclasses “Esquimó1” e “Esquimó 2” 20](#_Toc35418958)

[3.4.2.2. Subclasse “Pinguim” 21](#_Toc35418959)

[3.4.2.3. Subclasse PlataformaGelo 21](#_Toc35418960)

[3.4.2.3.1. Subclasse Plataforma\_Inicial 22](#_Toc35418961)

[3.4.2.3.2. Subclasse Plataforma\_Final 22](#_Toc35418962)

[3.4.2.4. Subclasse “Bandeira\_Start” 22](#_Toc35418963)

[3.4.2.5. Subclasse “Bandeira\_Finish” 23](#_Toc35418964)

[3.4.2.6. Subclasse Mar 23](#_Toc35418965)

1. **Introdução**

Este relatório tem por objetivo demonstrar como foram aplicados os conhecimentos adquiridos na unidade de Programação Orientada por Objetos para atingir a meta proposta pelos docentes, criar um jogo cooperativo usando a plataforma *Greenfoot* cujo tema fosse as Alterações Climáticas.

Este relatório explicará a ideia principal deste grupo tal como a forma de implementação e os procedimentos realizados para tal fim e a justificação pela qual decidiu-se implementá-los.

1. **Ideia Principal do Jogo**

A ideia principal do grupo foi criar um jogo dividido em 3 fases que têm que ser completados sequencialmente para ganhar o jogo. A primeira fase está relacionada com a emissão de gases tóxicos para a atmosfera, principalmente pelas grandes indústrias. A segunda fase está relacionada com os efeitos maliciosos de ditos gases na atmosfera do planeta, em particular a camada de ozono. A terceira e última fase está relacionada com o degelo das calotas polares, consequência das alterações climáticas provocadas pela poluição e não só.

1. **Procedimento e implementação do código**
   1. **Menus e Escolhas**
      1. **Classe *World***
         1. **Subclasse “MenuInicial”**

Este mundo permite aos utilizadores iniciar o jogo (depois de escolher os nomes e cores), redefinir os controlos ou sair do jogo.

Nesta subclasse tem-se implementado 2 construtores, um não recebe parâmetros enquanto que outro recebe um parâmetro booleano. Estes construtores só diferem na forma de definir a variável de instância reset: no construtor sem parâmetros esta variável é automaticamente true e no construtor com parâmetro booleano, reset é igual ao parâmetro passado.

Ambos os construtores criam um mundo de tamanho 1200 células por 700 células, em que cada célula mede 1 pixel por 1 pixel.

O método **prepare** instancia um objeto da classe “Play”, um objeto da classe “Options” e um objeto da classe “Exit”, todos nas posições devidas. O método prepare chama outro método chamado **resetStaticVariables** que, como o nome indica, vai repor aos valores “iniciais” as variáveis estáticas relacionadas com os jogadores.

Por fim, esta subclasse tem um método público **getMusica** que permite outros objetos parar a música de ambiente tocada.

### **3.1.1.2. Subclasse “Opções”**

Este mundo permite aos utilizadores alterarem os controlos que utilizarão no jogo.

O construtor desta subclasse define um mundo de tamanho igual ao tamanho do MenuInicial.

O método **prepare** os botões relacionados com a mudança dos controlos e os textos que explicam a tecla e a instrução relacionadas a cada controlo.

### **3.1.1.3. Subclasse “EscolhaNomes”**

Este mundo permite aos utilizadores registarem os seus nomes.

Nesta subclasse implementou-se um construtor que define as variáveis necessárias (nome é um string vazio, flag é true e recebeuNomeP1 é false) e cria um mundo das mesmas dimensões que o MenuInicial.

O método **prepare** instancia os objetos caixa, da classe Texto, que é um simples retângulo onde o objeto displayNome, também da classe Texto, mostra o nome que o utilizador está inserindo, um objeto Back, para retornar ao menu inicial, e outros textos informando o utilizador o que deve fazer. Ao utilizar este método no construtor, o mundo inicializa com os objetos nas coordenadas decididas automaticamente.

O método **escritaNome** é o método responsável por definir os nomes dos jogadores, estes nomes têm um limite de 10 carateres e o utilizador submete o seu nome carregando na tecla “enter”, como é explicado pelo mundo.

### **3.1.1.4. Subclasse “EscolherCor”**

Este mundo permite aos utilizadores escolherem as suas cores.

Nesta subclasse implementou-se um construtor que cria um mundo do mesmo tamanho que os anteriores e define que, após a escolha das cores, será apresentado o mundo HowToPlay com a explicação do primeiro nível.

O método **prepare** instancia os objetos relacionados com as cores (Azul, Amarelo, Verde, Vermelho e Preto) e um botão Back para regressar ao mundo EscolhaNomes.

### **3.1.1.5. Subclasse “HowToPlay”**

Este mundo simplesmente informa os utilizadores sobre o nível que jogarão ao clicar no botão Start.

Como nas subclasses anteriores, o construtor cria um mundo do tamanho anteriormente referido. Neste caso o método **prepare** demonstra a explicação (com recurso a objetos da classe Texto) e cria um botão da classe Start.

* + 1. **Classe Actor e Classe Menus**

Na classe Menus encontra-se o método **moveMouse** que recebe 2 imagens como parâmetros e alterna a imagem do objeto consoante o rato passe por cima do objeto, que irá ser chamado em todas as subclasses descritas abaixo. Também apresenta o método **playClick** que é chamada em todas as subclasses para reproduzir um som.

### **3.1.2.1. Subclasse “Play”**

Esta subclasse tem um método **clickMouse** que regista se o utilizador selecionou o objeto e cria um mundo da classe EscolhaNomes. O método **act** chama o método clickMouse e o método moveMouse herdado da superclasse Menus.

### **3.1.2.2. Subclasse “Options”**

Esta subclasse também tem um método **clickMouse**, só que quando regista que o utilizador selecionou o objeto cria um mundo da classe Opções e o método **act** é semelhante ao da subclasse anterior.

### **3.1.2.3. Subclasse “Exit”**

Esta subclasse é semelhante às anteriores, só que o método **clickMouse** também regista se o utilizador clicou a tecla “Escape” e para a simulação.

### **3.1.2.4. Subclasse “Controlos”**

Esta subclasse está encarregue de mudar os controlos consoante o utilizador deseje. O seu construtor recebe 2 inteiros, um designa qual o player a mudar o controlo e o outro designa qual controlo. O método **clickMouse** desta subclasse regista o click no objeto e, depois de despejar a última tecla digitada, fica à espera da tecla que será o novo controlo, e atualiza o texto que demonstra para o utilizador ficar informado. O método **inicio** insere o texto informando qual o controlo atual do jogador.

### **3.1.2.5. Subclasse “Texto”**

Esta subclasse é usada puramente para escrever texto no jogo. Tem o método **updateText**, que como o nome indica é usado para atualizar o texto.

### **3.1.2.6. Subclasse “Back”**

Esta subclasse é semelhante à subclasse “Exit”, pois o método **clickMouse** regista tanto o click do rato no objeto como regista o utilizador usar a tecla Escape para criar um novo mundo MenuInicial, sem fazer reset dos controlos.

* 1. **Jogo 1**

**Instruções:**

Nesta fase, o objetivo dos jogadores é destruir a máquina que está poluindo a atmosfera, dentro do limite de tempo de noventa segundos.

O jogo inicia com um jogador de cada lado do mundo. Cada jogador só pode permanecer no seu lado, não podendo ultrapassar a máquina e ir para o lado do outro jogador. Cada jogador tem 10 vidas, representadas por 5 corações. Os jogadores podem mover-se para a esquerda, direita, saltar, e também podem disparar, com as teclas predefinidas no jogo ou com aquelas que eles escolheram no menu “Controls”.

Para destruir a máquina os jogadores têm de atingi-la com as balas disparadas e ao mesmo tempo têm de se desviar do granizo na primeira etapa (que dura até a máquina atingir metade da sua vida) e depois na segunda etapa, têm de se desviar dos relâmpagos.

Antes de ser atingido por um relâmpago, o jogador é avisado através de um “alvo”, que aparece por debaixo dele, durante 3 segundos. Caso o relâmpago atinja o jogador, ele perde 2 vidas (1 coração).

Caso o granizo atinja o jogador, ele perde 1 vida.

Ao longo desta fase, para ajudar os jogadores, caem corações que lhes devolvem duas vidas.

Caso a máquina não seja destruída até o tempo acabar, os jogadores passam para a segunda fase se ambos ainda tiverem vidas. Caso contrário, como o jogo é cooperativo, basta apenas um jogador ficar sem vidas para o jogo terminar (game over).

Quanto mais vida a máquina tiver quando a fase acabar, maior será a dificuldade da fase seguinte.

* + 1. **Classe *World***

### **3.2.1.1 Subclasse “Jogo1”**

Este mundo corresponde à primeira fase do jogo. O seu construtor inicializa várias variáveis auxiliares, para a implementação dos vários métodos definidos nesta classe. Colocou-se as diferentes imagens de fundo em um array, definiu-se a ordem de pintura dos objetos no mundo, ou seja, que objetos sobrepõem quem, e chamou-se o método **prepare**.

Este último método, prepara o mundo de acordo com as características que pretendemos. Neste caso, o mundo é inicializado com vários objetos: o chão, os dois jogadores, a máquina e a barra com a sua vida, os nomes, vidas e pontuação de ambos os jogadores, o clock, e também inicializa os sons que serão utilizados (som ambiente e o som da chuva), e a variável midway, que será de extrema importância, uma vez que retorna o número correspondente a metade da vida da máquina, que delimita a primeira e a segunda etapa do jogo.

Esta classe possui quatro métodos, com o fim de definir a “queda” dos objetos no mundo: o **cairGranizo**, e o **cairNeve**, (usados na primeira etapa do jogo (Máquina.getVida > midway), com uma probabilidade de 5% e 10%, respetivamente), o **cairChuva** (usado na segunda etapa) com uma probabilidade de 20%, e o **cairVida**, usado em toda a fase, com uma probabilidade de apenas 0,2%.

Os métodos **aparecerNuvens** e **trocaFundo** são usados quando a vida da máquina atinge um valor igual ou inferior a midway, o primeiro método cria três objetos da classe “Nuvem” em cada lado do mundo simultaneamente na horizontal (para as nuvens parecerem mais escuras), e o segundo vai alterando o fundo consecutivamente, por outro mais escuro, até chegar ao fim do array, com o objetivo de parecer que o céu está a escurecer, por consequência do aparecimento das nuvens.

O método **getSomChuva**, retorna o som da chuva, da classe GreenfootSound.

É no método **invocaTarget**, que troca-se o som da etapa 1(som ambiente), para o som da etapa 2 (som da chuva), a partir do momento que a vida da máquina atinja a metade. Para além disto, tal como o nome indica, invoca um objeto da classe “Target” na posição atual do jogador em questão (através da instrução Greenfoot.getRandomNumber(2): se for 1, adiciona o objeto na posição do “Player1”, e se for 0 na posição do “Player2”), com uma probabilidade de 1,6%.

O método atualizaRelogio, controla o tempo do jogo (um minuto e meio), e vai alterando esse valor no mundo, sempre que a divisão inteira da variável contador (inicializada a 0) por 61, seja igual a zero. Para além disso, quando faltam dez segundos para acabar o tempo, as letras do clock passam a ser vermelhas, em vez de brancas. Quando o tempo acaba, todos os sons param, e altera o mundo atual, para um mundo da classe “Stage1Complete”.

Se um dos jogadores ficar sem vidas antes do tempo acabar, o método **gameOver**, pára todos os sons que estão a ser reproduzidos no momento, adiciona ao mundo um objeto da classe “GameOver” e outro da classe “Restart”, e reproduz o som de “gameOver”.

O método **geral**, chama vários dos métodos já referidos, quando a variável control é false: **cairGranizo**, **cairNeve**, **invocaTarget**, **atualizaRelogio**, **cairVida**, **aparecerNuvens** e **cairChuva**. Afora isso, este método também é responsável por atualizar a pontuação dos jogadores e chamar o método **trocaFundo**. Optou-se por colocar estes métodos, em um único só, para reduzir o número de métodos chamados no **act**. Deste modo, no **act** apenas é chamado o método **geral** e o **gameOver**.

### **3.2.2 Classe *Actor* >> Subclasse “Objetos” >> Subclasse “Jogos”**

### **3.2.2.1 Subclasse “ObjetosCaem”**

Nesta subclasse implementou-se os dois principais métodos que definem o comportamento dos objetos que “caem” no jogo 1. Sendo assim, todas as subclasses desta classe herdarão os mesmos métodos, sem ter que estar a defini-los em todas essas classes. Os métodos implementados foram: o método **ultrapassaLimite**, que define que quando qualquer um dos objetos que pertencem a esta classe, tocam no objeto da classe “Chão”, este é imediatamente removido do mundo, como forma de o jogo se aproximar um pouco mais da realidade, e também implementou-se o método **movimento**, que tal como o nome indica, define a rapidez e a direção com que os objetos se movimentam no mundo.

### **3.2.2.1.1 Subclasses Granizo, Gota e Neve**

Nestas três subclasses, apenas faz-se uma chamada no **act** dos dois métodos herdados da superclasse “ObjetosCaem”. No jogo 1, utiliza-se o granizo e a neve na primeira etapa e as gotas na segunda etapa.

### **3.2.2.1.2 Subclasse “Vida”**

Esta subclasse é muito semelhante às anteriores. No **act** chama-se o método **ultrapassaLimite**, porém esta classe tem um movimento mais lento e vertical no mundo, portanto definiu-se um método específico para ele chamado **movimentoVida**, e fez-se a sua chamada no **act**.

### **3.2.2.2 Subclasse “Target”**

No construtor desta classe definiu-se várias variáveis auxiliares do tipo inteiro (count2 e contador, inicializadas ambas a zero) e uma variável estática do tipo inteiro inicializada a 5 (TIMER). Para além disto, como a imagem do “Target” irá alternar entre com brilho e sem brilho, implementou-se um método **switchImage** que faz a troca da imagem sempre que o resto da divisão entre o count2 e o TIMER for igual a zero.

Como este objeto serve para avisar os jogadores que estão prestes a ser atingidos por um relâmpago, esta classe possui um método para esse fim, intitulado de **conta**. Esse método recebe como parâmetro um objeto da classe “Target”, e faz a contagem do tempo que este objeto permanecerá no mundo, com o auxílio da variável contador. Assim sempre que o contador chegar ao número 140, este retorna a zero, remove o “Target” e adiciona um novo objeto da classe "Relâmpago" no mundo.

### **3.2.2.3 Subclasse “Relâmpago”**

Esta subclasse é utilizada apenas na segunda etapa do jogo 1. Inicialmente começou-se por declarar uma variável de instância do tipo GreenfootImage na forma de array, com o propósito de reduzir o número de linhas de código necessárias para declarar todas as imagens, uma vez que

o relâmpago é a combinação de três imagens que vão mudando consecutivamente, para parecer um relâmpago real.

No construtor desta classe, recorreu-se a um ciclo for, para colocar cada imagem do relâmpago em um índice distinto no array. Para além disto, declarou-se uma variável de instância do tipo GreenfootSound para simular o som de um relâmpago real, e alterou-se o volume para 35.

O método que faz a troca de imagens é o método **switchImage**, com o auxílio das variáveis do tipo inteiro declaradas: indice, count e TIMER (inicializadas a zero, zero, e oito, respetivamente). Assim, quando o resto da divisão de count e TIMER for igual a zero, a imagem atual é trocada pela do índice seguinte (quando o índice é igual ao tamanho do array, este retorna a zero), enquanto que o count é inferior à multiplicação do TIMER pelo tamanho do array. Depois disso, o objeto é removido do mundo. Este é o único método chamado no **act**.

### **3.2.2.4 Subclasse “Nuvem”**

Esta classe, assim como a anterior, é apenas usada na segunda etapa e serve essencialmente para marcar melhor a passagem da primeira etapa para a segunda.

O construtor desta classe recebe um parâmetro do tipo inteiro “sentidoMovimento”, e dentro do construtor declarou-se duas variáveis auxiliares, também de tipo inteiro (aux e sentido), sendo uma inicializada a zero e a outra terá o mesmo valor do parâmetro, respetivamente.

No **act** faz-se a chamada a apenas um método: **movimentoNuvem**, querecebe a variável “sentido”. Este método tem apenas dois if’s, e no final é incrementada a variável aux. No primeiro if, enquanto a variável aux for inferior a 105 (para a nuvem não continuar se movimentando), a nuvem move-se na horizontal consoante o sentido que esta recebe, pois como o movimento é definido por uma multiplicação (sentido \* 3), caso este método receba como parâmetro um número negativo, o movimento da nuvem também será no sentido negativo e vice-versa. O segundo if serve apenas para fazer com que a nuvem se movimente um pouco de vez em quando para a esquerda ou direita, consoante um número Random.

### **3.2.2.5 Subclasse “Máquina”**

A máquina, ao longo do jogo 1, permanece sempre no mesmo lugar no mundo, até ser “destruída” pelos jogadores.

Primeiramente, definiu-se as variáveis do tipo int: vida, score, contador e indice (todas inicializam a zero, exceto a vida, que inicializa a 500). Implementou-se um array de imagens de 13 elementos, para simular a explosão da máquina, e no construtor colocou-se essas mesmas imagens no array através de um ciclo for.

No **act** são chamados dois métodos: **libertaGas** e **maquinaDestruida**. Como já tinha sido referido anteriormente, a máquina “liberta gás”, sendo assim implementou-se um método que faz essa simulação. O método **libertaGas** adiciona objetos da classe “Gás”, de forma aleatória dentro de determinados limites, para parecer que o gás está a sair da chaminé da máquina. Já o método **maquinaDestruida**, sempre que a vida de algum dos jogadores for inferior ou igual a zero e o resto da divisão inteira do contador (incrementado no final do método) por três for zero (para trocar a imagem mais lentamente), o som da chuva para, e toca o som da explosão quando o índice do array é zero (para tocar o som apenas uma vez), depois a imagem é trocada por aquela que tem um índice superior, e por fim quando o array chegar ao fim, é removida a máquina e é adicionado ao mundo um objeto da classe “Stage1Complete”.

E por fim, implementou-se dois métodos muito simples: o método **getVida**, que devolve a vida atual da máquina, e o método **tiraVida**, que recebe um inteiro “valor”, e devolve a subtração da vida por esse número.

### **3.2.2.5.1 Subclasse “VidaMáquina”**

Para mostrar a vida da máquina ao longo do jogo, recorreu-se a uma barra “healthbar” de cor roxa, delimitada por uma linha branca, em que o retângulo roxo vai ficando mais estreito à medida que a máquina vai perdendo vida. Para criar esta barra, implementou-se o método **atualiza** e utilizou-se vários métodos da classe GreenfootImage.

Inicialmente definiu-se as constantes do tipo int: HEALTHBARCOMPRIMENTO, HEALTHBARALTURA, inicializadas a 1000 e 15, respetivamente, health (com o mesmo valor da variável vida da classe “Máquina”, e percentagemDeVida que corresponde à divisão de HEALTHBARCOMPRIMENTO por health.

No método **atualiza** começa-se por mudar a imagem da classe para uma imagem transparente com mais dois pixéis de largura e comprimento que as constantes HEALTHBARCOMPRIMENTO e HEALTHBARALTURA, posteriormente desenhou-se, a branco, o contorno do retângulo (através dos métodos **setColor** e **drawRect** do GreenfootImage), com menos um pixel de largura e comprimento que a imagem anterior, e no final preencheu-se a barra, que nos informará a vida da máquina, com a cor roxa (através dos métodos **setColor** e **fillRect** do GreenfootImage) dentro do limite branco. A largura da barra roxa é calculada fazendo-se a multiplicação da percentagemDeVida pela health. Como a variável percentagemDeVida é estática, ela permanece com o mesmo valor do início ao fim (neste caso, 2), pois só faz a divisão apenas uma vez, e a health, como não é estática, vai se alterando à medida que a máquina perde vida. Este método é chamado no **act**.

### **3.2.2.6 Subclasse “Chão”**

Esta subclasse não tem nenhum método, uma vez que apenas serve de referência para outras classes (tais como: “ObjetosCaem” e “Jogo1”), pois faz a distinção entre aquilo que é “terra” e o que é “céu” no jogo.

### **3.2.2.7 Subclasse “Gás”**

O “Gás” está presente não só no jogo 1, mas também no jogo 2, e em ambos os jogos este objeto tem o mesmo comportamento.

Sendo assim, implementou-se dois métodos, e chamou-se ambos no **act**: o método **ultrapassaLimite**, cuja função é apenas remover o objeto quando este chega ao limite do mundo. E o método **movimentoGas**, que tal como o nome nos diz, define o movimento do gás. Para o gás não ter um movimento uniforme “para cima”, recorreu-se a uma variável do tipo Random, para que o gás se mova entre -5 e 5 pixéis na horizontal, com uma probabilidade de 10%.

### **3.2.3 Classe *Actor***

### **3.2.3.1 Subclasse “Players”**

Nesta subclasse estão definidos métodos para os três jogos. Sendo assim, nesta parte, iremos apenas abordar e explicar as variáveis e métodos que influenciam apenas o jogo 1. Os restantes métodos serão explicados, ao longo do relatório.

Esta subclasse é responsável por definir vários métodos importantes para o funcionamento e animação dos jogadores:

Existem dois métodos **perdeVidas**, um deles recebe como parâmetro um objeto da classe “Player1” (P1) e o outro um objeto da classe “Player2” (P2), uma vez que cada jogador tem uma dinâmica diferente no jogo, dependendo das pessoas que os estão a controlar. Este método tem como função controlar os ganhos e perdas de vidas dos jogadores. Deste modo, se um objeto da classe “Granizo” tocar em um objeto da classe “Player1” ou “Player2”, o jogador perde meia vida, o objeto “Granizo” é removido, é retirado 50 pontos ao jogador, e o som que determina que o jogador foi atingido é tocado. Se um dos “players” for atingido por um objeto da classe “Relâmpago”, caso a variável booleana tocandoRelampago seja false, é lhe retirado uma vida e cem pontos, a variável é alterada para true e o som do relâmpago é tocado, caso contrário a variável passa a ser false novamente. Já se os “Players” tocarem em um objeto “Vida”, estes ganham duas vidas e 25 pontos, a “Vida” é removida e o som que o jogador ganhou uma vida é tocado.

O método **animarMove** é responsável por animar o movimento dos jogadores. Para parecer que os “Players” estão se movendo, definiu-se um array de imagens, que alternadas imitam o movimento humano. Deste modo, o método recebe como parâmetro esse array de imagens (animacao), e procede á alteração da imagem sempre que a variável auxiliar contador é igual a quatro, retomando a zero depois disso.

Assim como o método **animarMove**, o método **animarMorte**, serve para animar o movimento dos jogadores, porém este método é apenas usado quando a vida do jogador é igual ou inferior a zero, para simular a “morte” do mesmo. A forma de animar o movimento é semelhante ao do método anterior, contudo este método, recebe como parâmetro, além do array de imagens, uma variável do tipo boleano (andandoParaEsquerda) que informa se o jogador está andando para a esquerda (true) ou direita (false). Originalmente, as imagens que simulam a morte dos jogadores são todas imagens em que o jogador cai virado para a direita, deste modo não faz sentido usar essas imagens quando o jogador “morre” virado para a esquerda. Para resolver esse problema, utilizou-se o método do GreenfootImage **mirrorHorizontally**, que espelha a imagem original horizontalmente, quando a variável andandoParaEsquerda é true.

Os métodos **playAtingido** e **playVida** são responsáveis por tocar um som quando o jogador é atingido por uma bola de granizo e por uma vida, respetivamente.

Para não existirem conflitos entre as teclas inseridas e os outros controlos definiu-se o método **podeMudar**, que recebe como parâmetros três strings, correspondentes à tecla que o jogador está a pressionar, e as outras strings os controlos que estão a ser executados nesse momento pelo “Player1” e pelo “Player2”, e devolve um valor boleano. Este método é utilizado na classe “Controlos”.

### **3.2.3.1.1 Subclasse “Player1” e “Player2”**

Estas duas subclasses têm exatamente os mesmos métodos, apenas trocou-se onde diz na subclasse “Player1”, P1, na subclasse “Player2” diz P2, e vice-versa (P1 e P2 são objetos da classe “Player1” e “Player2”, respetivamente).

No construtor destas classes, colocou-se as imagens referentes ao movimento e morte dos jogadores num array de imagens de acordo com a cor escolhida no início do jogo.

Nesta subclasse, para cumprir-se o requisito do encapsulamento, implementou-se vários métodos simples, que servem para informar o estado atual de cada um dos jogadores. No “Player1”, os métodos foram: **getP2Morreu** (devolve true se o “Player2” morreu), **setP2Morreu** (recebe como parâmetro um boleano e altera o estado de vida do “Player2”, consoante o valor desse parâmetro), **getNome** (devolve a string com o nome do jogador), **setNome** (altera o nome do jogador consoante a string que recebe como parâmetro), **getScore** (devolve a pontuação do jogador), **resetScore** (altera a pontuação atual para zero), **adicionaScore** (adiciona e subtrai pontuação, ao valor inserido), **getAndandoParaEsquerda** (devolve true se o jogador está andando para a esquerda), **getControls** (retorna o array de strings com os controlos), **setControls** (altera o movimento do jogador, com base na tecla que está a pressionar), **setColor** (altera a cor atual dos “Players”), **getColor** (retorna a cor do jogador), **getNumeroVidas** (retorna o número de vidas do jogador), **resetNumVidas** (altera o número de vidas para o valor inicial) e **adicionaNumeroVidas**, que adiciona e subtrai vidas ao jogadores pelo valor introduzido como parâmetro, tendo em atenção que, caso o jogador tenha mais de oito vidas, não é adicionada mais vida, pois o máximo é dez. Deste modo, nenhuma outra classe acede de forma direta às variáveis desta classe.

O método **move** controla todos os movimentos do jogador, e apenas funciona quando ambos os jogadores têm pelo menos uma vida. É constituído por vários if’s, correspondentes a cada uma das teclas pressionadas (up, right e left). Se o jogador pressionar a tecla up e a variável podeSaltar for true, as variáveis boleanas podeSaltar e saltou passam a ser false e true, respetivamente. De seguida, se saltou for true, o salto é executado usando o método **jump**. Se for pressionada a tecla left e o jogador ainda estiver virado para a direita, é efetuada a alteração da imagem para um espelho da mesma, depois é chamado o método animaMove e alterada a variável andandoParaEsquerda e a posição do jogador. O mesmo acontece se pressionarmos a tecla right (e não estiver tocando na máquina), só que a imagem só é alterada se o jogador estiver virado para a esquerda. E por fim, quando nenhuma tecla está a ser pressionada, a imagem atual volta a ser a do índice 0 do array.

Como a variável tempoJump é igual ao valor da GRAVIDADE (15), sempre que o método **jump** é chamado o primeiro if é executado até esta ser igual a zero, depois disso volta a ter o mesmo valor inicial, e a variável saltou passa a ser false.

Enquanto que o método **jump** trata do salto, o método **queda**, faz com que o jogador volte para o chão, sempre que saltou for false e o jogador não estiver tocando no “Chão”. Depois disso, podeSaltar passa a ser true novamente.

O método **disparar**, cria uma nova instância da classe “Bala” no lugar onde se encontra o jogador, e faz com que ela se mova de acordo com o sentido para que o jogador está virado (esquerda sentido negativo e direita sentido positivo).

No **act** são chamados os métodos **queda**, **move**, **disparar** e **perdeVidas**, e impôs-se uma condição, para quando a vida for inferior ou igual a zero, chamar o método **animarMorte.**

### **3.2.3.1.1.1 Subclasses “Vida\_player1” e “Vida\_player2”**

Estas subclasses, assim como as anteriores, são basicamente idênticas.

No construtor destas subclasses é carregado as imagens correspondentes às vidas dos jogadores, em um array. Cada imagem corresponde a um índice no array, que por sua vez está relacionada com um determinado número de vida que o jogador possui. Sendo assim, o método **vidaPlayer1**, chamado no **act**, (ou **vidaPlayer2**, no caso do “Player2”), é encarregado de ir substituindo a imagem das vidas, consoante a informação do número de vidas de cada jogador.

### **3.2.3.1.2 Subclasse “Bala”**

Esta subclasse tem dois construtores: um para o “Player1” e outro para o “Player2”, exatamente com as mesmas variáveis e valores (apenas troca-se o P1 pelo P2 e vice-versa).

O método **disparo** controla todo o movimento da bala. Se a variável mudaOrientacao for false (coisa que só acontece uma vez), a imagem é alterada consoante a posição do jogador e mudaOrientacao passa a ser true. Posteriormente, é definido o movimento da bala, com o auxílio da constante VELOCIDADE.

O método **desapareceLimite** retorna true, se a “Bala” está em algum dos limites do mundo.

Para saber-se que a bala atingiu a máquina, implementou-se o método **atingiuMáquina**, que para além de nos informar se a bala tocou na máquina (devolve um boleano), ele também adiciona score aos jogadores, recorrendo à variável P1Disparou ou P2Disparou, que nos informa se algum dos jogadores disparou. De seguida, retira vida à máquina, enquanto é superior a zero.

No final são descritos dois métodos: **getP1Disparou** e **getP2Disparou**, que devolvem o valor da variável P1Disparou e P2Disparou, respetivamente.

No act, faz-se uma chamada do método **disparo**, e define-se um if, que caso algum dos métodos **desapareceLimite** e **atingiuMáquina** retornar true, o objeto desta classe é removido.

* 1. **Jogo 2**

**Instruções:**

Nesta fase, o objetivo principal é evitar que os gases tóxicos, libertados pela máquina na primeira fase, não atinjam a camada de ozono, durante 2 minutos.

O jogo inicia com duas naves com a mesma vida com que os jogadores terminaram a fase anterior. As naves têm as mesmas cores que os jogadores escolheram inicialmente. Elas podem mover-se na horizontal, para esquerda e para a direita, entre os limites do mundo.

Ao longo do jogo são lançados misseis teleguiados com a mesma cor da nave que eles pretendem atingir. Ao ser atingido por um míssil (de qualquer cor), o jogador perde duas vidas e 100 pontos.

Assim como na primeira fase, a camada de ozono tem uma barra a indicar a sua vida que irá diminuir caso os gases a atinjam.

Caso a vida da camada de ozono chegue a 0 ou uma das naves seja destruída, o jogo termina.

No decorrer do jogo, aparecem corações para os jogadores recuperarem vidas.

Quanto menor for a vida da camada de ozono no final desta fase, maior será a dificuldade da última fase.

* + 1. **Classe *World***
       1. **Subclasse “Jogo2”**

O construtor deste nível cria um mundo com o tamanho referido anteriormente, define o relógio em 2 minutos, toca o som ambiente do nível, inicializa a variável quantoGas que será mencionada mais à frente e invoca o método prepare que inicializa todos os objetos necessários.

No método **prepare**, os objetos inicializados são as naves dos jogadores (*Nave1* e *Nave2*), os displays das vidas (*Vida\_player1* e *Vida\_player2*), a camada de ozono (*CamadaOzono*) e a sua vida (*VidaCamadaOzono*) e os textos que visualizam o relógio, os nomes dos jogadores e a sua pontuação.

Este mundo tem o método **atualizaRelogio** já mencionado no Jogo1, o método **libertarGases** que, como o nome indica, invoca objetos da classe Gas com uma probabilidade 3/(100 – quantoGas), de modo a que a quantidade de Gas libertado seja tanto maior quanto mais vida a máquina acabou no nível 1.

O método **vidas** invoca objetos da classe Vida\_jogo2 com uma probabilidade de 1/500. O método **disparaMissil**, como o nome indica, invoca objetos da classe Missil com uma certa probabilidade e, dependendo do número aleatório gerado, o míssil tem como alvo a nave do jogador 1 ou a nave do jogador 2.

O método **gameOver** recebe como parâmetros a vida da CamadaOzono e as vidas dos jogadores pois se algum destes parâmetros chegar a 0, os jogadores perdem o jogo.

O método **geral** invoca os métodos mencionados anteriormente, exceto o gameOver, enquanto a variável control é falsa, que indica que os jogadores ainda não perderam o jogo e é chamado no **act** do mundo, junto com o método gameOver.

* + 1. **Classe *Actor***
       1. **Subclasses “Nave1” e “Nave2”**

Estas subclasses são quase idênticas, a única diferença é que Nave1 é subclasse de Player1 e Nave2 é subclasse de Player2.

Os construtores destas subclasses carregam a imagem da nave dependendo da cor que os jogadores escolheram e carrega as imagens relacionadas à explosão da nave, caso aconteça, num array denominado explosão.

As subclasses têm dois métodos: **moveNave**, que é o método relacionado com o movimento da nave, esta só pode mover-se para a esquerda e direita enquanto um dos jogadores tem vida, e o método **removeGas** que, como o nome indica, remove objetos da classe Gas quando a nave toca em dito objeto e adiciona 10 pontos ao jogador que removeu o Gas.

No método **act** é chamado os 2 métodos da subclasse e os métodos **perdeVidas** e **naveDestruida**, quando o jogador perdeu toda a sua vida, herdados da superclasse Players.

* + - 1. **Subclasse “Gas”**

Esta subclasse é simples, não apresenta um particular construtor e apresenta 2 métodos, ambos chamados no **act**.

O método **ultrapassaLimite** é simples, elimina o objeto se este chegou ao limite do mundo, só útil no Jogo1, e o método **movimentoGas** move o Gas para cima uma célula e possivelmente para a esquerda ou direita, um número aleatório de células.

* + - 1. **Subclasse “CamadaOzono”**

No construtor desta subclasse é definida a vida da camada (125) e definida a transparência da imagem da camada.

O método **act** só chama o método **mudarTransparencia** que diminui a vida e redefine a transparência da camada de acordo com a vida da camada, se um objeto da classe Gas toca na camada e remove o objeto da classe Gas.

A subclasse também tem um método estático **getVida** para obter o valor da vida da camada de ozono.

* + - 1. **Subclasse “VidaCamadaOzono”**

Esta subclasse é praticamente idêntica à subclasse VidaMáquina, a única diferença as dimensões da barra e a cor da barra.

* + - 1. **Subclasse “Missil”**

O construtor da subclasse Missil recebe como parâmetros um inteiro valor, que define a variável de instância limite, que define até qual y o míssil é teleguiado, um boolean alvoENave2, que indica se o alvo do míssil será a nave do jogador 2 ou a nave do jogador 1, e referências às duas naves no mundo. O construtor também define a cor do míssil dependendo da cor do alvo e carrega as imagens relativas à explosão do míssil num array.

Esta subclasse apresenta o método **teleguiado**, alvo de overloading de modo a que receba ou uma referência a um objeto da Nave1 ou da Nave2, que aponta o míssil ao alvo e move o míssil 2 células, se este míssil ainda estiver abaixo do limite.

O método **chegouAoFim** remove o míssil se este chegar a um limite do mundo, sem que este expluda e o método **atingiuAlvo** regista se o míssil está a tocar uma das Naves e retira meia vida a esse jogador e retira 100 pontos a esse jogador; também define a variável explosao como true de modo a que se possa chamar o método **animaExplosao** que, como o nome indica, simplesmente anima a explosão do míssil e, no fim da explosão, remove o objeto.

No método **act**, se explosao é false, os métodos teleguiado, com o alvo como parâmetro, atingiuAlvo e chegouAoFim são chamados; caso contrário, se o míssil atingiu uma nave, é animada a sua explosão.

* 1. **Jogo 3**

Nesta última fase, o objetivo principal é chegar à meta, saltando entre plataformas de gelo que se movem continuamente para a esquerda.

O jogo inicia com os jogadores (esquimós) numa plataforma inicial que se encontra parada até as próximas plataformas chegarem perto dos jogadores.

Nesta fase, os jogadores podem mover-se para a esquerda, direita e saltar.

Ao longo do jogo aparecem pinguins em cima das plataformas de gelo. Os jogadores para salvá-los têm ambos de tocar no pinguim ao mesmo tempo, ganhando assim 25 pontos cada. Caso contrário, ao se aproximarem do limite esquerdo do mundo, os pinguins caem da plataforma e cada jogador perde 100 pontos.

A largura das plataformas depende da fase anterior.

A fase é concluída quando ambos os jogadores chegam à plataforma final. Se um dos jogadores cair no mar, o jogo termina (game over).

* + 1. **Classe *World***
       1. **Subclasse “Jogo3”**

O construtor desta fase cria um mundo com o mesmo tamanho dos mundos anteriores, toca uma música, são inicializadas as variáveis inteiras **contadorMar**, **altura\_anterior**, **conta\_plataformas**, **pinguinsSalvos** como também as variáveis booleanas control, P1Chegou e P2Chegou. De seguida irá ser explicada a função de cada uma.

No método prepare são inicializados dois objetos da classe *Mar*, as personagens dos dois jogadores (*Esquimó1* e *Esquimó2*), uma plataforma de gelo da classe *Plataforma\_Inicial*, uma bandeira que indica a posição inicial dos jogadores (*Bandeira\_Start*), como também textos que indicam os nomes e scores de cada jogador e o número de pinguins salvos pelos jogadores.

O método **incrementarPinguimSalvo** permite que seja feito um incremento da variável pinguinsSalvos, inicialmente a 0, que irá contar o número de pinguins salvos pelos jogadores.

O método **resetPinguimSalvo** permite que a contagem de pinguins salvo recomece do inicio, ou seja, a partir de 0.

Os métodos **setP1Chegou** e s**etP2Chegou** permitem que sejam atualizadas as variáveis P1Chegou e P2Chegou, que inicialmente eram “false”. Caso o jogador 1 chegue à plataforma final, a variável P1Chegou é atualizada para “true”. O mesmo acontece para o caso do jogador 2, mas desta vez sendo a variável P2Chegou atualizada.

O método **invocarPlataformas** é responsável por invocar as plataformas de gelo que os jogadores utilizarão para chegar à plataforma final. Neste método é criada constantemente um inteiro que é um valor entre 0 e 2. Caso este número seja 0, é criada uma plataforma a uma altura de 300. Caso o número seja 1, é criada uma plataforma a uma altura de 400. E, caso o número seja 2, é criada a uma altura de 500. No máximo são criadas 25 plataformas de gelo, tendo cada uma um respawn de 2 segundos. Quando o número máximo de plataformas é alcançado, é criada um objeto da classe Plataforma\_Final e um da classe Bandeira\_Finish.

O método **invocarMar** permite a criação constante de objetos da classe Mar.

O método **gameOver** é semelhante aos métodos gameOver das classes Jogo1 e Jogo2. Recebe como parâmetros a vida do jogador 1 e a vida do jogador 2. Se algum destes parâmetros chegar a 0, os jogadores perdem o jogo, aparecendo um aviso de “Game Over”, como também um botão da classe Restart que permite reiniciar o jogo.

O método **vitoria** recebe como argumentos as variáveis P1Chegou e P2Chegou. Caso estas variáveis sejam “true”, significa que o objetivo dos jogadores foi completado, sendo assim inicializado um mundo da classe Victory.

O método **geral** recorre à chamada dos métodos anteriormente referidos. Enquanto a variável control seja “false”, ou seja, os jogadores continuarem vivos, são criadas plataformas pelo método invocarPlataformas e é verificado se os jogadores já chegaram à plataformaFinal. Este método é chamado no método act, junto com o método invocarMar.

* + 1. **Classe *Actor***
       1. **Subclasses “Esquimó1” e “Esquimó 2”**

Tal como as classes “Player1” e “Player2”, estas duas subclasses têm exatamente os mesmos métodos. Estes métodos diferem apenas nos métodos herdados que são chamados. No caso do Esquimó1 é feita a chamada de métodos herdados da superclasse Player1. Enquanto que, no caso do Esquimó2, é feita a chamada de métodos herdados da superclasse Player2.

No construtor destas classes, colocou-se as imagens referentes ao movimento dos jogadores num array de imagens de acordo com a cor escolhida no início do jogo.

Nestas subclasses é visível a ocorrência de ***overriding***. Ambas têm o método **move**, responsável pelo movimento dos jogadores. As únicas diferenças destes métodos com os métodos move das classes Player1 e Player2 são que os esquimós movem-se ligeiramente mais rápido e não existe nenhuma verificação se as classes estão a tocar algum outro objeto de uma classe distinta, que possa limitar o seu movimento.

Também, o método jump, responsável pela subida no salto dos jogadores, é idêntico aos das classes Player1 e Player2.

O método **queda** é responsável pela descida no salto dos jogadores, evita que os jogadores fiquem a meio do bloco e regista se os jogadores já chegaram à plataforma final. Neste método, caso o jogador não esteja em cima de uma plataforma da classe PlataformaGelo, a velocidade da queda aumenta ao longo do tempo. Se o jogador estiver a tocar numa plataforma, é colocado em cima dela, permitindo que possa saltar. Se essa plataforma for da classe Plataforma\_Final, é registado que o jogador chegou à meta.

No método **act** de cada uma das subclassesé feita a chamada dos métodos move e queda, como também do método cair herdado da superclasse Player1 e Player2.

* + - 1. **Subclasse “Pinguim”**

Esta subclasse tem um construtor onde é criado um array com imagens para criar uma animação do objeto, tocado o som emitido por um pinguim quando o objeto aparece no mundo, criado o som que irá tocar quando o pinguim for salvo pelos jogadores e, três variáveis, sendo duas do tipo inteiro, inicializadas a zero, e uma do tipo booleana, inicializada como “false”.

O método **animar** é responsável pela animação do pinguim quando ele se encontra em cima de uma plataforma de gelo.

O método **salvado** verifica se ambos os jogadores estão a tocar o pinguim. Caso isto aconteça, é tocado um som e o pinguim desaparece, indicando que o pinguim foi salvo com sucesso. Ao ser salvo, ambos os jogadores recebem 25 pontos e é atualizado o número de pinguins salvos visível no mundo da classe Jogo3.

O método **emCimaPlataforma** verifica se o pinguim se encontra em cima de uma plataforma da classe PlataformaGelo. Se o pinguim não estiver em contacto com a plataforma, irá cair, atualizando a variável caindo para true, que é utilizada pelo método seguinte.

O método **caindo** altera a imagem atual do pinguim e faz com que o pinguim rode, dando assim um animação de queda do objeto, caso a variável caindo seja true.

O método **desaparece** é responsável por remover o objeto quanto este toca o limite inferior do mundo. Se isto acontecer, significa que os jogadores não conseguiram salvar o pinguim, fazendo com que cada jogador perca 100 pontos.

* + - 1. **Subclasse PlataformaGelo**

O construtor desta subclasse recebe um inteiro que irá determinar o tamanho das plataformas que irão ser criadas durante o jogo, diminuindo a largura da imagem da plataforma de gelo. É também, chamado o método podeCriarPinguim que será explicado mais tarde.

O método **derreter** faz com que as plataformas criadas transmitam aos jogadores uma ideia de que estão a derreter. Para isso, são criados objetos da classe Gota e é diminuída a transparência do objeto ao longo que se aproxima do limite esquerdo do mundo. Também, quando a plataforma se encontra muito próxima do limite esquerdo do mundo, começa a mover-se para baixo e ligeiramente a rodar para a esquerda, dando a ideia que o objeto caiu.

O método **podeCriarPinguim** decide se a plataforma de gelo irá criar um objeto da classe Pinguim quando for criado, atualizando a variável pinguim do tipo booleana, que quando é “true” indica que a plataforma pode criar um objeto da classe Pinguim. A probabilidade da plataforma poder criar um pinguim é de 20%.

No método **invocaPinguim** verifica se a variável pinguim anteriormente referida é “true”. Caso isto aconteça, é criado um pinguim numa posição aleatória em cima da plataforma.

O método **desaparecer** faz com que o objeto seja removido do mundo quando este chega ao limite esquerdo ou inferior do mundo. É também, tocando um som de “splash” que dá a ideia de que a plataforma caiu no mar.

No método **act** é feita a chamada de todos estes métodos e do método move, com exceção do método podeCriarPinguim que é chamado no construtor da subclasse.

* + - * 1. **Subclasse Plataforma\_Inicial**

O construtor desta subclasse inicializa a zero uma variável inteira que irá controlar o duração do tempo que a plataforma terá de permanecer parada até começar o seu movimento, é alterada a largura da plataforma de acordo com a vida que a camada de ozono tinha quando o nível anterior acabou. Quanto maior for a vida da camada de ozono, menor será a largura das plataformas.

No método **act** são chamados os métodos **desaparecer**, herdado da superclasse PlataformaGelo, e **permanecerParado** que faz com que a plataforma só se comece a mover para a esquerda quando a variável, que inicialmente era zero, for superior a 650.

* + - * 1. **Subclasse Plataforma\_Final**

O construtor desta subclasse inicializa a zero uma variável inteira que irá controlar o duração do tempo que esta plataforma se irá mover, e dá uma imagem ao objeto.

No método **act** é chamado apenas o método **permanecerParado**, que faz com que o objeto se mova para a esquerda até a variável que inicialmente era zero, chegar a 99.

* + - 1. **Subclasse “Bandeira\_Start”**

No construtor desta subclasse é também inicializada a zero um variável inteira que irá ter a mesma função do que a variável criada no construtor da classe Plataforma\_Inicial.

O método **permanecerParado** é idêntico ao da classe Plataforma\_Inicial.

O método **desaparecer** faz com que o objeto seja removido do mundo quando chega ao limite esquerdo do mesmo.

No método **act** são chamados os dois métodos anteriores.

* + - 1. **Subclasse “Bandeira\_Finish”**

No construtor desta subclasse é , mais uma vez, inicializada a zero uma variável inteira com a mesma função do que a variável criada no construtor da classe Plataforma\_Final.

No método **act** é chamado o método **permanecerParado** que é idêntico ao método da classe Plataforma\_Inicial.

* + - 1. **Subclasse Mar**

Esta subclasse serve apenas para dar a ideia de *side-scrolling*.

Os objetos desta classe movem-se constantemente para a esquerda através do método **move**, e são removidos do mundo que estes chegam ao limite esquerdo do mesmo, através do método **remove**.

1. **Requisitos**
   1. **Jogo**
      1. **2 ou mais jogadores que colaboram**

Todos os 3 níveis estão feitos para que sejam possíveis dois 2 colaborarem ao mesmo tempo.

No primeiro nível os 2 jogadores têm de colaborar para destruir a máquina, onde os jogadores são controlados pelos controlos escolhidos.

No segundo nível os 2 jogadores têm de colaborar para manter a camada de ozono intacta, evitando que os gases tóxicos cheguem a esta, movendo-se para a esquerda e direita para “apanhar” os gases.

No terceiro e último nível, ambos os jogadores têm de chegar ao fim e salvar, em conjunto, os pinguins.

Em qualquer dos níveis, se um jogador morre, o jogo acaba, visto ser um jogo colaborativo.

* + 1. **Score com a pontuação atualizada em tempo real**

Em qualquer dos níveis a pontuação dos jogadores é visualizada em tempo real. Decidimos implementar o valor que guarda o score dos jogadores nas classes Player1 e Player2, para ser mais fácil atualizá-lo consoante os eventos que acontecessem, e nos mundos dos níveis era simplesmente chamado o método updateText para os jogadores visualizarem a sua pontuação atualizada em tempo real.

* + 1. **Indicador de vida/energia/tempo ou semelhante para cada jogador**

Os primeiros dois níveis são cronometrados, o primeiro dura 90 segundos enquanto que o segundo dura 120, logo em ambos os níveis é mostrado aos jogadores um indicador do tempo restante, se o tempo termina o nível acaba. Nestes dois primeiros níveis também é indicado aos jogadores quanta vida estes possuem, atualizada em tempo real.

No último nível, como o objetivo dos jogadores é chegar ao fim e cair ao mar é morte instantânea, não se achou útil demonstrar um indicador de tempo ou indicador das vidas dos jogadores.

* + 1. **Modificação do aspeto do mundo e dos jogadores durante o jogo**

No primeiro nível, existem duas etapas, onde a mudança das etapas é sinalizada pelo escurecimento do fundo e a aparência de nuvens no céu de onde chove e cai relâmpagos. Os jogadores quando se movem são animados para transmitir a noção de movimento e, se um jogador morre, a morte também é animada.

No segundo nível, sempre que a camada de ozono é atingida por um gás tóxico a sua transparência diminui. Se um míssil atinge uma nave a explosão do míssil também é animada e, se um jogador perde toda a vida, a sua nave “explode”, através de uma animação.

No último nível as plataformas de gelo ficam mais transparentes ao longo da distância que percorrem, para dar o efeito de “derreter”; o movimento dos jogadores é animado como no primeiro nível e, se caem ao mar, ficam transparentes até desaparecer e, por fim, os pinguins acenam aos jogadores para que estes o salvem.

* + 1. **Indicação da pontuação obtida ao finalizar o jogo**

No fim de cada nível a pontuação conjunta dos dois jogadores é mostrada, através do mundo Stage1Complete ou Stage2Complete e no fim do jogo (após os 3 níveis), a pontuação conjunta é mostrada através do mundo Victory.

* 1. **Programação Orientada por Objetos**
     1. **Inicialização de objetos usando os construtores**

O uso de construtores para inicializar objetos é algo presente em quase todas as classes, um exemplo é a classe Missil onde através do construtor é definido a cor do míssil e o alvo, tendo em base os parâmetros passados (ver subclasse Missil para mais informação).

* + 1. **Herança de métodos com um mínimo de 2 níveis além de Actor**

A herança de métodos é o conceito de uma subclasse herdar os métodos não privados da sua superclasse e da superclasse da sua superclasse e assim por diante, de forma a evitar a reutilização de código e facilitar implementação.

Este objetivo também se tornou algo comum no código, por exemplo na classe Menus encontra-se o método moveMouse responsável por alterar a imagem do objeto consoante o rato do utilizador passe em cima do objeto e este método é chamado em quase todas as suas subclasses.

* + 1. **“Overriding” de métodos**

*Overriding* de um método é a sua redefinição numa subclasse, sobrepondo-se à definição herdada da superclasse.

Efetuamos este conceito com o método **move** e **jump**, ambos presentes em Player1, mas como precisávamos que estes métodos no 3º nível fossem mais específicos, redefinimo-los na subclasse Esquimó1, assim efetuando *overriding* dos métodos herdados de Player1.

* + 1. **“Overloading” de métodos**

*Overloading* é o processo de ter vários métodos com o mesmo nome, só que com diferentes assinaturas, nomeadamente, listas de parâmetros diferentes.

No nosso código pode-se ver este conceito na classe Players, onde o método **perdeVidas** tanto pode receber um objeto do tipo Player1, Player2, Nave1 ou Nave2 como parâmetro. Também se pode verificar este conceito na classe Missil, onde o método **teleguiado** pode receber como parâmetro um objeto do tipo Nave1 ou Nave2.

* + 1. **Encapsulamento**

Este conceito é uma das regras de ouro de POO: as variáveis de instância devem ser sempre privadas, ou seja, a alteração destas variáveis só ocorre na classe onde está declarada ou através de métodos “set”.

Sendo uma regra de ouro, é possível verificar este conceito no nosso código em quase todas as classes, por exemplo na classe Máquina a variável que guarda a vida da máquina é privada, só podendo ser vista através do método público getVida.

1. **Conclusão**

Concluindo, os objetivos pretendidos neste projeto foram alcançados, e o desenvolvimento deste jogo proporcionou aos alunos um maior conhecimento sobre a criação de jogos básicos e foi uma boa maneira de introduzir a Programação Orientada a Objetos.

A plataforma Greenfoot provou ser bastante funcional para a implementação deste projeto e para a aprendizagem de POO.

O grupo acha o resultado final muito satisfatório.