UNIVERSIDADE da MADEIRA

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

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.

2. 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.

3. Procedimento e implementação do código

3.1. Menus e Escolhas

3.1.1. Classe *World*

3.1.1.1 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.

3.1.1.6. Subclasse "Stage1Complete" e "Stage2Complete"

Estes dois mundo são semelhantes. A única diferença é os argumentos dos métodos chamados.

Na subclasse Stage1Complete, o construtor cria um mundo do mesmo tamanho dos mundos anteriores, é colocada a imagem do mesmo, começa a tocar a música da classe MenuInicial através da chamada dos métodos **getMusica** da classe MenuInicial e, é feita a chamada dos métodos **setJogo1**, da classe IndividualScore, que será referida de seguida, com "true" como argumento. Também, é feita a chamada do método **mudarJogo** da classe HowToPlay, sendo "2" o argumento, indicando que o nível seguinte é o jogo 2.

Na subclasse Stage2Complete, o construtor faz exatamente o mesmo do que o construtor da subclasse Stage1Complete, mas no caso do método **setJogo1**, o argumento é "false" e, no caso do método **mudarJogo** o argumento é "3". Assim, é indicado que o nível seguinte é o jogo 3.

3.1.1.7. Subclasse "IndividualScore"

Este mundo serve para os jogadores poderem ver o score obtido por cada um deles.

No construtor desta classe é criado um mundo, mais uma vez, com as mesmas dimensões, é colocada a imagem do mundo e chamado o método prepare.

O método **prepare** criar dois objetos da classe Texto que dão a informação do nome de cada jogador seguido do score obtido pelo mesmo, e de um objeto da classe GoBack, que permite os jogadores voltem aos mundos Stage1Complete e Stage2Complete.

O método **getJogo1** retorna o valor da variável jogo1, que se for "true" significa que o jogo completado anteriormente foi o jogo 1.

O método **setJogo1** recebe uma variável booleana que permite alterar o valor do jogo1.

3.1.2. 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 **início** 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.

3.1.2.7. Subclasse “Stage”

Esta classe serve apenas para organizar melhor as classes utilizadas nos mundos Stage1Complete, Stage2Complete, IndividualScore e HowToPlay. As suas subclasses são todas semelhantes à classe “Exit”, tendo cada uma caraterísticas especificas.

3.1.2.7.1. Subclasse “GoBack”

O método **clickMouse** regista o click do rato no objeto e verifica se o jogo concluído anteriormente foi o jogo 1 ou o jogo 2, através da chamada do método getJogo1 da classe IndividualScore, criando assim um mundo da classe Stage1Complete ou da classe Stage2Complete. Este método e o método moveMouse herdado da superclasse são chamados no método **act** da subclasse.

3.1.2.7.2. Subclasse “Menu”

O método **clickMouse** regista o click do rato no objeto e cria um mundo da classe MenuInicial.

3.1.2.7.3. Subclasse “NextLevel”

O método **clickMouse** mouse regista o click do rato no objeto e cria um mundo da classe HowToPlay.

3.1.2.7.4. Subclasse “PlayersScore”

O método **clickMouse** regista o click do rato no objeto e cria um mundo da classe IndividualScore.

3.1.2.7.5. Subclasse “Start”

O método **clickMouse** regista o click do rato no objeto, parando assim a música atual e verificando o valor retornado pelo método jogoAtual da classe HowToPlay. Caso o valor seja “1”, é criado um mundo da classe Jogo1. Caso o valor seja “2”, é criado um mundo da classe Jogo2. E, caso o valor seja “3”, é criado um mundo da classe Jogo3.

3.2. 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 uma vida.

Ao longo desta fase, para ajudar os jogadores, caem corações que lhes devolvem uma vida.

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.

3.2.1. 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**, que recebe 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, 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 uma 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 duas vidas 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.

O método **cair**, que pode receber como parâmetro um objeto Esquimó1 ou Esquimó2 e o array de imagens da animação, é responsável por registar a queda do objeto ao mar, sinalizando a morte do esquimó e atualizando as variáveis P1Morreu e P2Morreu para true e a variável P1Chegou, no caso de passar um Esquimó1, ou a variável P2Chegou, no caso de passar um Esquimó2, a false, pois estando morto já não pode registar que chegou.

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 9 vidas, é adicionada uma 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 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.

3.3. 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 uma vida 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.

3.3.1. Classe "*World*"

3.3.1.1. 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.

3.3.2. Classe "*Actor*"

3.3.2.1. 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.

3.3.2.2. 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.

3.3.2.3. 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.

3.3.2.4. Subclasse “VidaCamadaOzono”

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

3.3.2.5. 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 uma 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.

3.4. Jogo 3

Instruções:

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).

3.4.1. Classe "*World*"

3.4.1.1. 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 início, 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.

3.4.2. Classe "*Actor*"

3.4.2.1. 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 subclasses é feita a chamada dos métodos move e queda, como também do método cair herdado da superclasse Player1 e Player2.

3.4.2.2. 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 uma animação de queda ao 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 200 pontos.

3.4.2.3. 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 de 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.

3.4.2.3.1. Subclasse "Plataforma_Inicial"

O construtor desta subclasse inicializa a zero uma variável inteira que irá controlar a 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.

3.4.2.3.2. 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.

3.4.2.4. 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.

3.4.2.5. 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.

3.4.2.6. 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**.

4. Requisitos

4.1. Jogo

4.1.1. 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.

4.1.2. 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.

4.1.3. 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.

4.1.4. 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.

4.1.5. 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.

4.2. Programação Orientada por Objetos

4.2.1. 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).

4.2.2. 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.

4.2.3. "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.

4.2.4. "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.

4.2.5. 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.

5. 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 achou o resultado final muito satisfatório.

6. Anexos de código

6.1. Classes *World*

6.1.1. EscolhaNomes

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class EscolhaNomes extends World
{
   private Texto caixa;
   private Texto displayNome;
   private Texto header;
   private String nome;
   private boolean flag;
   private boolean recebeuNomeP1;

   public EscolhaNomes()
   {
      super(1200, 700, 1);
      GreenfootImage bg = new GreenfootImage("chooseName.png");
      bg.scale(getWidth(), getHeight());
      setBackground(bg);
      nome="";
      flag=true;
      recebeuNomeP1=false;
      prepare();
   }

   /**
    * Prepare the world for the start of the program.
    * That is: create the initial objects and add them to the world.
    */

   private void prepare()
   {
      //CRIA RETÂNGULO ONDE APARECE O NOME DO JOGADOR:
      int larg=getWidth();
      int alt=getHeight();
      caixa=new Texto();
```

```
        caixa.setImage(new GreenfootImage(500, 50));
        caixa.getImage().setColor(Color.BLACK);
        caixa.getImage().fillRect(0,0,500, 50);
        caixa.getImage().setColor(Color.WHITE);
        caixa.getImage().fillRect(2,2,496, 46);
        addObject(caixa, getWidth()/2, 3*getHeight()/4);

        //MOSTRA NOME A SER ESCRITO
        displayNome = new Texto();
        displayNome.getImage().clear();
        addObject(displayNome, getWidth()/2, 3*getHeight()/4);

        //TEXTO INFORMATIVO
        header=new Texto("Player 1,\nwrite your name",50, new Color(255,255,255));
        addObject(header, getWidth()/2, getHeight()/2);
        addObject(new Texto("Max size of 10 characters",30, new Color(222, 18,
195)),getWidth()/2,3*getHeight()/4-40);
        addObject(new Texto("Press enter to submit",30, new Color(222, 18,
195)),getWidth()/2,3*getHeight()/4+60);
    }

    public void act()
    {
        escritaNome();
    }

    /**
     * Método que vai receber as teclas introduzidas pelo utilizador, construindo o nome do jogador
e mostrando-o visualmente
     */
    private void escritaNome()
    {
        Greenfoot.getKey(); //discarda a última tecla pressionada
        String key=null;
        while (flag)
        {
            key = Greenfoot.getKey();
```

```
        if (key!=null){
            if(key.length()<2) //certifica que nao escreve "shift" ou "control" ou outra tecla não alfa-numerica
            {
                nome+=key;
            }
            else if (key.equals("space")) //meter espaço no nome
            {
                nome+=" ";
            }
            flag=false;
        }

    }
    if (key.equals("enter") || nome.length()>=10) //o utilizador finaliza o seu nome pressionando o enter ou se o tamanho do nome for igual (ou superior) a 10 carateres
    {
        displayNome.updateText(nome,displayNome,40, new Color(0,255,0));
        Greenfoot.delay(25);
        if(!recebeuNomeP1) //define o nome do primeiro jogador e permite receber o do segundo jogador
        {
            Player1.setNome(nome);
            displayNome.updateText("Player 2,\nwrite your name", header,50, new Color(255,255,255));
            nome="";
            recebeuNomeP1=true;
        }
        else //define o nome do segundo jogador e muda o mundo para escolher as cores dos jogadores
        {
            Player2.setNome(nome);
            Greenfoot.setWorld(new EscolherCor());
        }
    }
    else if (key.equals("backspace")) //permite apagar uma letra do nome
    {
```

```
            if(nome.length()>0)
            {
                nome=nome.substring(0, nome.length()-1);
            }
        }
        displayNome.updateText(nome, displayNome, 40, Color.BLACK);
        flag=true;
    }
}
```

6.1.2. EscolherCor

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class EscolherCor extends World
{
    public EscolherCor()
    {
        super(1200, 700, 1);
        GreenfootImage bg = new GreenfootImage("Earth.png");
        bg.scale(getWidth(), getHeight());
        setBackground(bg);
        prepare();
        HowToPlay.mudarJogo(1); //indica que é para indicar as instruções relevantes à primeira fase
    }

    private void prepare()
    {
        int larg=getWidth();
        int alt=getHeight();
        addObject(new Cor(0),larg/6,2*alt/3);
        addObject(new Cor(1),2*larg/6,2*alt/3);
        addObject(new Cor(2),3*larg/6,2*alt/3);
        addObject(new Cor(3),4*larg/6,2*alt/3);
        addObject(new Cor(4),5*larg/6,2*alt/3);
        Texto text =new Texto(Player1.getNome()+",\npick your colour",60, new Color(255,255,255));
        addObject(text,larg/2,alt/3);
```

```
            addObject(new Back(), larg-100, alt-50);
        }
    }
```

6.1.3. HowToPlay

```
    import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

    public class HowToPlay extends World
    {
        private Texto info;
        private static int jogo;
        public HowToPlay()
        {
            super(1200, 700, 1);
            GreenfootImage bg = new GreenfootImage("Earth.png");
            bg.scale(getWidth(), getHeight());
            setBackground(bg);
            prepare();
        }

        /**
         *  Métodos que permitem determinar qual explicação mostrar aos jogadores dependendo do jogo que vão jogar
         */

        public static void mudarJogo(int aux){
            jogo = aux;
        }

        public static int jogoAtual(){
            return jogo;
        }

        public void prepare(){
            switch (jogo) //instruções fase do jogo (fase 1,2 ou 3) que será jogado
            {
                case 1:
```

```
            info = new Texto("In this stage, your task is to destroy the machine \nthat is polluting the atmosphere, but be careful! \nDodge the falling hail and lightning or the game will be over!",45, new Color(222, 18, 195));
            break;
            case 2:
            info = new Texto("Now that you have survived those harsh conditions, climb aboard\nyour spaceships and prevent the toxic gases that machine released \nfrom completely destroying the ozone layer! But it won't be that \neasy, you'll have to dodge the guided missiles being fired at you!",45, new Color(222, 18, 195));
            break;
            case 3:
            info = new Texto("For this final stage all you need to do is navigate through the ice \nplatforms from start to finish and save the penguins for \nbonus score, but be careful because \none wrong move and all the work will go to waste!",45, new Color(222, 18, 195));
            break;
        }
        addObject(info, getWidth()/2, getHeight()/2);
        addObject(new Start(), getWidth()/2,516);

    }
}
```

6.1.4. IndividualScore

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class IndividualScore extends World
{
    private Texto scoreP1;
    private Texto scoreP2;
    private GoBack back;
    private static boolean jogo1;
    public IndividualScore()
    {
        super(1200, 700, 1);
        GreenfootImage bg = new GreenfootImage("individualScore.png");
        bg.scale(getWidth(), getHeight());
        setBackground(bg);
```

```
        prepare();
    }

    public void prepare(){
        scoreP1 = new Texto(Player1.getNome()+": "+Player1.getScore(),30, new
Color(255,255,255));
        addObject(scoreP1, getWidth()/2 , 260);
        scoreP2 = new Texto(Player2.getNome()+": "+Player2.getScore(),30, new
Color(255,255,255));
        addObject(scoreP2, getWidth()/2 , getHeight()/2 +10);

        back = new GoBack();
        addObject(back,getWidth()/2,468);
    }

    public static boolean getJogo1()
    {
        return jogo1;
    }

    public static void setJogo1(boolean x)
    {
        jogo1=x;
    }
}
```

6.1.5. Jogo1

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Jogo1 extends World
{
    private boolean control;
    private int tempo;
    private String escreverClock;
    private Texto scoreP1, scoreP2, clock;
    private Player1 P1;
    private Player2 P2;
    private static GreenfootSound somAmbiente;
```

```
        private static GreenfootSound somChuva;
        private int contador, midway, auxNuvem, auxFundo;
        private final int TAMANHOTEXTO=45;
        private GreenfootImage[] fundos;
        private int indice;

        public Jogo1()
        {
           super(1200, 700, 1);
           control =false;
           contador=0;
           tempo = 90;
           escreverClock = "1:30";

           auxNuvem = 0;
           setPaintOrder(Texto.class,GameOver.class,
Restart.class,VidaMáquina.class,Vida_player1.class,Vida_player2.class,Nuvem.class,Relâmpago
.class, Máquina.class, Gas.class,Gota.class );

           auxFundo = -1;
           fundos= new GreenfootImage[9];
           for(int i=0; i <fundos.length;i++)
           {
              fundos[i]=new
GreenfootImage("jogo1_backgrounds/jogo1_background_"+(i+1)*5+".jpg");
           }
           indice=0;
           prepare();
        }

        private void prepare()
        {
           somAmbiente = new GreenfootSound("winter.mp3");
           somChuva = new GreenfootSound("rain.mp3");
           somAmbiente.playLoop();
           somAmbiente.setVolume(30);
```

```
        Chão chao= new Chão();
        addObject(chao,getWidth()/2,getHeight()-chao.getImage().getHeight()/2);

        Máquina máquina = new Máquina();
        addObject(máquina,getWidth()/2,getHeight()-chao.getImage().getHeight()-
máquina.getImage().getHeight()/2);

        P1 = new Player1();
        addObject(P1,getWidth()/10,getHeight()-chao.getImage().getHeight()-
P1.getImage().getHeight()/2+2);
        P2 = new Player2();
        addObject(P2,9*getWidth()/10,getHeight()-chao.getImage().getHeight()-
P2.getImage().getHeight()/2+2);

        addObject(new VidaMáquina(),getWidth()/2,40);
        addObject(new Vida_player1(), 285/2, 125);
        addObject(new Vida_player2(), getWidth()-285/2, 125);

        clock = new Texto(escreverClock,TAMANHOTEXTO, new Color(255,255,255));
        addObject(clock, getWidth()/2,125);

        scoreP1 = new Texto(""+Player1.getScore(),TAMANHOTEXTO, new
Color(255,255,255));
        addObject(scoreP1, 285/2,175);
        scoreP2 = new Texto(""+Player2.getScore(),TAMANHOTEXTO, new
Color(255,255,255));
        addObject(scoreP2, getWidth()-285/2,175);

        addObject(new Texto(Player1.getNome(),TAMANHOTEXTO-10, new
Color(255,255,255)),285/2,80);
        addObject(new Texto(Player2.getNome(),TAMANHOTEXTO-10, new
Color(255,255,255)),getWidth()-285/2,80);

        midway=máquina.getVida()/2;
    }

    public void act(){
```

```
            gameOver(P1.getNumeroVidas(),P2.getNumeroVidas());
            geral();
        }

        private void geral(){
            if (!control) //quando um dos jogadores morrerem, o jogo acaba e já não efetua estes métodos
            {
                cairGranizo();
                cairNeve();
                invocaTarget();
                cairVida();
                cairChuva();
                aparecerNuvens();
                atualizaRelogio();
            }
            scoreP1.updateText(""+Player1.getScore(), scoreP1, TAMANHOTEXTO, scoreP1.getCor());
            scoreP2.updateText(""+Player2.getScore(), scoreP2, TAMANHOTEXTO, scoreP2.getCor());
            trocaFundo();
        }


        public static GreenfootSound getSomChuva()
        {
            return somChuva;
        }

        /**
         * Método que verifica se um dos jogadores morreu, se isto acontecer, os jogadores perderam o jogo
         */
        private void gameOver(int vidaJogador1, int vidaJogador2){
            if ((vidaJogador1 <=0 || vidaJogador2 <= 0) && !control ){
                addObject(new GameOver(),getWidth()/2,getHeight()/2);
                addObject(new Restart(),getWidth()/2,getHeight()/2 +150);
```

```
            somAmbiente.stop();
            somChuva.stop();
            Greenfoot.playSound("gameOver.mp3");
            control =true;
        }
    }

    /**
     * Método que atualiza o tempo que falta para o nível acabar e, quando o tempo acaba,
     * avança para o mundo onde os jogadore podem ver o score total e individual
     */
    private void atualizaRelogio()
    {
        contador++;
        if(contador%61==0)
        {
            tempo--;
            if(tempo%60<10)
            {
                escreverClock = "" + tempo/60 + ":0" + tempo%60;
            }
            else
            {
                escreverClock = "" + tempo/60 + ":" + tempo%60;
            }
        }
        if (tempo <=10 && contador%61<30)
        {
            clock.updateText(escreverClock,clock,TAMANHOTEXTO, Color.RED);
        }
        else
        {
            clock.updateText(escreverClock,clock,TAMANHOTEXTO, Color.WHITE);
        }
        if(tempo==0)
        {
            somAmbiente.stop();
```

```
            somChuva.stop();
            Greenfoot.setWorld(new Stage1Complete());
        }
    }

    /**
     * Métodos que criam granizo e neve, respetivamente, enquanto a máquina tem mais de metade da sua vida inicial
     */
    private void cairGranizo()
    {
        if (Máquina.getVida() > midway )
        {
            if (Greenfoot.getRandomNumber(100)<7)
            {
                addObject(new Granizo(), Greenfoot.getRandomNumber(getWidth()-1),0);
            }
        }

    }

    private void cairNeve()
    {
        if (Máquina.getVida() > midway ){
            if (Greenfoot.getRandomNumber(100)<10)
            {
                addObject(new Neve(), Greenfoot.getRandomNumber(getWidth()-1),0);
            }
        }
    }

    /**
     * Método que cria o alvo onde o relampago cairá e a chuva, respetivamente, após a máquina ter perdido metade da sua vida.
     * Os relâmpagos cairão na posição atual do jogador para incentivar o movimento e dificultar o jogo
     */
```

```
        private void invocaTarget()
        {
           if(Máquina.getVida() <=midway)
           {
              somAmbiente.stop();
              if(!somChuva.isPlaying()) //evita que o som se repita
              {
                 somChuva.playLoop();
                 somChuva.setVolume(40);
              }
              int prob = Greenfoot.getRandomNumber(2);
              if (Greenfoot.getRandomNumber(250)<4){
                 Target target = new Target();
                 if (prob == 1 && !getObjects(Player1.class).isEmpty())
                 {
                    addObject(target, P1.getX(), 620); //o x é o do jogador para o jogador não ficar
parado
                 }
                 else if (prob == 0 && !getObjects(Player2.class).isEmpty())
                 {
                    addObject(target, P2.getX(), 620); //o x é o do jogador para o jogador não ficar
parado
                 }
              }
           }
        }

        /**
         * Método que faz com que caiam quando a máquina tem menos de metade da vida
         */
        private void cairChuva(){
           if (Máquina.getVida() <= midway ){
              if (Greenfoot.getRandomNumber(100)<20)
              {
                 addObject(new Gota(), Greenfoot.getRandomNumber(getWidth()-1),0);
              }
           }
```

```
    }

    /**
     * Método que cria corações que os jogadores podem apanhar para recuperarem vida que possam ter perdido
     */

    private void cairVida()
    {
       if (Greenfoot.getRandomNumber(500)<1)
       {
          addObject(new Vida(), Greenfoot.getRandomNumber(getWidth()-1),0);
       }

    }

    /**
     * Método que sinaliza que a máquina perdeu metade da sua vida e a "fase" dos relâmpagos começará
     */
    private void aparecerNuvens(){
       if (Máquina.getVida() <= midway && auxNuvem == 0){
          //3 objetos de cada lado para as nuvens ficarem mais escuras
          addObject(new Nuvem(1), 0,2);
          addObject(new Nuvem(1), 0,2);
          addObject(new Nuvem(1), 0,2);
          addObject(new Nuvem(-1), getWidth(),10);
          addObject(new Nuvem(-1), getWidth(),10);
          addObject(new Nuvem(-1), getWidth(),10);
          auxNuvem++;
       }
    }

    /**
     * Método que escurece o fundo quando a máquina perdeu metade da sua vida
     */
    private void trocaFundo()
```

```
    {
        if (Máquina.getVida() <= midway)
        {
            if (auxFundo%12==0 && auxFundo <=96)
            {
                setBackground(fundos[indice]);
                indice++;
            }
            auxFundo++;
        }
    }
}
```

6.1.6. Jogo3

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Jogo3 extends World
{
    private int contadorMar;
    private static int pinguinsSalvos;
    private final int ALTURA1=300;
    private final int ALTURA2=400;
    private final int ALTURA3=500;
    private Texto scoreP1, scoreP2,pinguins;
    private final int TAMANHOTEXTO=45;
    private int contador;
    private int altura_anterior;
    private int conta_plataformas;
    private boolean control;
    private static boolean P1Chegou, P2Chegou;
    private final int NUMPLATAFORMASPARAGANHAR=30;
    private Esquimó1 esq1;
    private Esquimó2 esq2;
    private static GreenfootSound somAmbiente;
    public Jogo3()
    {
        super(1200, 700, 1);
```

```
        contadorMar=0;
        prepare();
        contador = 150;
        altura_anterior=0; //o valor 0 não interfere com a escolha da altura na primeira interação, apenas serve para inicializar a variável
        conta_plataformas=0;
        control = false;
        P1Chegou=false;
        P2Chegou=false;
        pinguinsSalvos = 0;
    }

    private void prepare()
    {

        setPaintOrder(Texto.class, GameOver.class, Restart.class, Player1.class, Player2.class, Pinguim.class, PlataformaGelo.class);
        Mar mar = new Mar();
        addObject(mar,603,getHeight() -30);
        Mar mar2 = new Mar();
        addObject(mar2,1012,getHeight() -30);

        esq1 = new Esquimó1();
        addObject(esq1,390,300);
        esq2 = new Esquimó2();
        addObject(esq2,410,300);

        scoreP1 = new Texto(""+Player1.getScore(),TAMANHOTEXTO, new Color(0,0,0));
        addObject(scoreP1, 285/2, 125);
        scoreP2 = new Texto(""+Player2.getScore(),TAMANHOTEXTO, new Color(0,0,0));
        addObject(scoreP2, getWidth()-285/2, 125);
        addObject(new Texto(Player1.getNome(),TAMANHOTEXTO, new Color(0,0,0)),285/2,80);
        addObject(new Texto(Player2.getNome(),TAMANHOTEXTO, new Color(0,0,0)),getWidth()-285/2,80);

        addObject(new Plataforma_Inicial(),400,ALTURA2);
```

```
        Texto desenhoPinguim = new Texto();
        desenhoPinguim.setImage("penguinImage.png");
        addObject(desenhoPinguim, getWidth()/2 - 20, 90);
        pinguins = new Texto(""+pinguinsSalvos,TAMANHOTEXTO, new Color(0,0,0));
        addObject(pinguins,getWidth()/2 +20, 90);

        addObject(new Bandeira_Start(),400,348);
        somAmbiente = new GreenfootSound("snowField.mp3");
        somAmbiente.playLoop();
        somAmbiente.setVolume(30);
    }

    public void act(){
        geral();
        invocarMar();

    }

    public void geral(){
        if(!control) //como anteriormente, se um dos jogadores morre, estes métodos param de
ser chamados
        {
            invocarPlataformas();
            vitoria(P1Chegou,P2Chegou);
        }
        gameOver(esq1.getNumeroVidas(), esq2.getNumeroVidas());
        pinguins.updateText(""+pinguinsSalvos,pinguins,TAMANHOTEXTO,
pinguins.getCor());
        scoreP1.updateText(""+Player1.getScore(),  scoreP1,  TAMANHOTEXTO,
scoreP1.getCor());
        scoreP2.updateText(""+Player2.getScore(),  scoreP2,  TAMANHOTEXTO,
scoreP2.getCor());
    }

    /**
     * Métodos que incrementa o contador de pinguins salvos
```

```
     */
    public static void incrementarPinguimSalvo(){
       pinguinsSalvos++;
    }

    /**
     * Método que faz reset ao contador de pinguins salvos
     */
    public static void resetPinguimSalvo(){
       pinguinsSalvos = 0;
    }

    /**
     * Métodos que permitem na classe de Esquimó1 e 2 registar que estes chegaram à plataforma final
     */
    public static void setP1Chegou(boolean x)
    {
       P1Chegou=x;
    }

    public static void setP2Chegou(boolean x)
    {
       P2Chegou=x;
    }

    /**
     * Método que permite invocar as plataformas de gelo que os jogadores utilizarão para chegar ao fim
     */
    public void invocarPlataformas()
    {
       int random =Greenfoot.getRandomNumber(3);
       if (contador%150==0 && conta_plataformas<NUMPLATAFORMASPARAGANHAR+1)
       {
```

```
            while(altura_anterior==2 && random ==0) //evita que os jogadores estejam na plataforma mais baixa e apareca uma na altura mais alta, o que é um salto impossivel para os jogadores
            {
               random = Greenfoot.getRandomNumber(3);
            }
            switch (random)
            {
               case 0:
               addObject(new PlataformaGelo(-conta_plataformas),getWidth()-1,ALTURA1);
               break;
               case 1:
               addObject(new PlataformaGelo(-conta_plataformas),getWidth()-1,ALTURA2);
               break;
               case 2:
               addObject(new PlataformaGelo(-conta_plataformas),getWidth()-1,ALTURA3);
               break;
            }
            conta_plataformas++;
            altura_anterior=random;
         }

         if(contador%150==0 && conta_plataformas==NUMPLATAFORMASPARAGANHAR) //verifica que os jogadores ja percorreram plataformas suficientes
         {
            addObject(new Plataforma_Final(),getWidth()-1,640);
            addObject(new Bandeira_Finish(),getWidth()-1,530);
         }
         contador++;
      }

      /**
       * Método que inicia os objetos da classe mar que dão a aparência de side scrolling
       */

      public void invocarMar(){
```

```
            contadorMar++;
            if(contadorMar == 200){
                addObject(new Mar(),getWidth()-10,getHeight() -30);
                contadorMar=0;
            }
        }

        /**
         * Método que verifica se um dos jogadores caiu e morreu, o que indica que ambos os jogadores, como equipa, perderam o jogo
         */
        private void gameOver(int vidaJogador1, int vidaJogador2){
            if ((vidaJogador1 <=0 || vidaJogador2 <= 0) && !control ){
                P1Chegou = false;
                P2Chegou = false;
                addObject(new GameOver(),getWidth()/2,getHeight()/2);
                addObject(new Restart(),getWidth()/2,getHeight()/2 +150);
                somAmbiente.stop();
                Greenfoot.playSound("gameOver.mp3");
                control =true;
            }
        }

        /**
         * Método que verifica se ambos os jogadores chegaram à plataforma final, o que indica que GANHARAM!!
         */
        private void vitoria(boolean P1chegou, boolean P2chegou)
        {
            if(P1chegou && P2chegou)
            {
                somAmbiente.stop();
                Greenfoot.setWorld(new Victory());
            }
        }
    }
```

6.1.7. MenuInicial

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class MenuInicial extends World
{
   private static GreenfootSound musica;
   private boolean reset;

   public MenuInicial(boolean resetControlos)
   {
      super(1200,700, 1);
      reset=resetControlos;
      GreenfootImage bg = new GreenfootImage("Earth.png");
      musica = new GreenfootSound("music.mp3");
      bg.scale(getWidth(), getHeight());
      setBackground(bg);
      prepare();
      Greenfoot.setSpeed(50);
   }

   /**
    * Construtor só usado no inicio da simulação, para definir os controlos que os jogadores podem mudar
    */
   public MenuInicial()
   {
      super(1200,700, 1);
      reset=true;
      GreenfootImage bg = new GreenfootImage("Earth.png");
      musica = new GreenfootSound("music.mp3");
      bg.scale(getWidth(), getHeight());
      setBackground(bg);
      prepare();
      Greenfoot.setSpeed(50);
   }

   private void prepare()
   {
```

```
        addObject(new Play(),getWidth()/2,349);
        addObject(new Options(),getWidth()/2,420);
        addObject(new Exit(),getWidth()/2,462);
        resetStaticVariables();
    }

    /**
     * Método que faz um reset às variáveis static, nomeadamente aquelas relacionadas com os jogadores como os seus controlos, score e número de vidas
     */
    private void resetStaticVariables(){
        if(reset)
        {
            Player1.setControls(0, "w");
            Player1.setControls(1, "a");
            Player1.setControls(2, "d");
            Player1.setControls(3, "f");
            Player2.setControls(0, "up");
            Player2.setControls(1, "left");
            Player2.setControls(2, "right");
            Player2.setControls(3, "0");
        }
        Player1.resetScore();
        Player2.resetScore();
        Player1.resetNumVidas();
        Player2.resetNumVidas();
        Jogo3.resetPinguimSalvo();
    }

    /**
     * Método que certifica que a música só toca quando o jogo está a ser executado e não repete
     */
    public void started()
    {
        musica.playLoop();
    }
```

```
    public static GreenfootSound getMusica(){
        return musica;
    }

    public void stopped()
    {
        musica.stop();
    }
}
```

6.1.8. Opções

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)


public class Opções extends World
{
    private final int TAMANHOTEXTO=50;
    public Opções()
    {
        // Create a new world with 600x400 cells with a cell size of 1x1 pixels.
        super(1200,700, 1);
        GreenfootImage bg = new GreenfootImage("Earth.png");
        bg.scale(getWidth(), getHeight());
        setBackground(bg);
        prepare();
    }

    private void prepare()
    {
        int larg = getWidth();
        int comp = getHeight();
        addObject(new Texto("Player 1",TAMANHOTEXTO, new Color(255,255,255)), larg/3, comp/6);
        addObject(new Texto("Player 2",TAMANHOTEXTO, new Color(255,255,255)), (2*larg)/3, comp/6);
        addObject(new Texto("Jump",TAMANHOTEXTO, new Color(255,255,255)), larg/8, 2*comp/6);
```

```
            addObject(new Texto("Left",TAMANHOTEXTO, new Color(255,255,255)), larg/8,
3*comp/6);
            addObject(new Texto("Right",TAMANHOTEXTO, new Color(255,255,255)), larg/8,
4*comp/6);
            addObject(new Texto("Shoot",TAMANHOTEXTO, new Color(255,255,255)), larg/8,
5*comp/6);

            addObject(new Controlos(1,0), larg/3, (2*comp)/6);
            addObject(new Controlos(1,1), larg/3, (3*comp)/6);
            addObject(new Controlos(1,2), larg/3, (4*comp)/6);
            addObject(new Controlos(1,3), larg/3, (5*comp)/6);
            addObject(new Controlos(2,0), (2*larg)/3, (2*comp)/6);
            addObject(new Controlos(2,1), (2*larg)/3, (3*comp)/6);
            addObject(new Controlos(2,2), (2*larg)/3, (4*comp)/6);
            addObject(new Controlos(2,3), (2*larg)/3, (5*comp)/6);
            addObject(new Back(), larg-100, comp-50);
        }
    }
```

6.1.9. Stage1Complete

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Stage1Complete extends World
{
    private int finalScore;
    private Texto score;
    public Stage1Complete()
    {
        super(1200, 700, 1);
        GreenfootImage bg = new GreenfootImage("Stage1Complete.png");
        bg.scale(getWidth(), getHeight());
        setBackground(bg);
        prepare();
        IndividualScore.setJogo1(true);
        MenuInicial.getMusica().playLoop();
        HowToPlay.mudarJogo(2);
    }
```

```
    /**
     * Prepare the world for the start of the program.
     * That is: create the initial objects and add them to the world.
     */
    private void prepare()
    {
        finalScore = Player1.getScore() + Player2.getScore();
        score = new Texto("SCORE: "+finalScore,45, new Color(255,255,255));
        addObject(score, getWidth()/2 , getHeight()/2 +15);

        Menu menu = new Menu();
        addObject(menu,500,468);

        NextLevel nextLevel = new NextLevel();
        addObject(nextLevel,700,468);

        PlayersScore players_score = new PlayersScore();
        addObject(players_score,600,468);
    }
}
```

6.1.10. Stage2Complete

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)


public class Stage2Complete extends World
{
    private int finalScore;
    private Texto score;
    public Stage2Complete()
    {
        super(1200, 700, 1);
        GreenfootImage bg = new GreenfootImage("Stage2Complete.png");
        bg.scale(getWidth(), getHeight());
        setBackground(bg);
        prepare();
        IndividualScore.setJogo1(false);
        MenuInicial.getMusica().playLoop();
```

```
        HowToPlay.mudarJogo(3);
    }

    private void prepare()
    {
        Menu menu = new Menu();
        addObject(menu,500,468);

        NextLevel nextLevel = new NextLevel();
        addObject(nextLevel,700,468);

        PlayersScore players_score = new PlayersScore();
        addObject(players_score,600,468);

        finalScore = Player1.getScore() + Player2.getScore();
        score = new Texto("SCORE: "+finalScore,45, new Color(255,255,255));
        addObject(score, getWidth()/2 , getHeight()/2 +15);
    }
}
```

6.1.11. Victory

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Victory extends World
{
    private int finalScore;
    private Texto score;

    public Victory()
    {
        super(1200, 700, 1);
        Greenfoot.playSound("victory.mp3");
        prepare();
    }

    public void prepare(){
        finalScore = Player1.getScore() + Player2.getScore();
        score = new Texto("FINAL SCORE: "+finalScore,60, new Color(0, 255, 255));
```

```
        addObject(score, getWidth()/2 , getHeight()/2 +150);
    }
}
```

6.2. Classes *Actor*, subclasses de "Jogos"

6.2.1. Bandeira_Finish

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Bandeira_Finish extends Jogos
{
    private int contador;

    public Bandeira_Finish()
    {
        contador = 0;
    }

    public void act()
    {
        permanecerParado();
    }


    /**
     * Método que move a bandeira de acordo com o movimento da plataforma final, que é onde esta bandeira estará
     */
    public void permanecerParado()

    {
        if (contador<100)
        {
            move(-1);
        }
        contador++;

    }
}
```

6.2.2. Bandeira_Start

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Bandeira_Start extends Jogos
{
   private int contador;
   public Bandeira_Start()
   {
      contador =0;

      GreenfootImage imagemAtual=getImage();

      imagemAtual.scale(imagemAtual.getWidth()-(125-CamadaOzono.getVida())/5,
imagemAtual.getHeight());

   }

   public void act()
   {
      desaparecer();
      permanecerParado();
   }


   /**
    * Método que move a bandeira de acordo com o movimento da plataforma inicial, onde
a bandeira está
    */
   public void permanecerParado()

   {
      if (contador >650)
      {
         move(-1);
      }
      contador++;
```

```
    }

    /**
     * Método que faz com que a bandeira desapareça do mundo, assim que esta toque no limite esquerdo do mundo (X=0)
     */
    public void desaparecer()
       {

       if (getX() == 0){
          getWorld().removeObject(this);
       }
    }
}
```

6.2.3. CamadaOzono

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class CamadaOzono extends Jogos
{
   private GreenfootImage imageCamada;
   private static int vida;

   public CamadaOzono(){
      imageCamada= getImage();
      vida=125;
      imageCamada.setTransparency(2*vida+5);
   }

   public void act()
   {
      mudarTransparencia();
   }

   public static int getVida(){
      return vida;
   }
```

```
        /**
         * Método que regista a perda de vida da camada de ozono e torna esta mais transparente de acordo com a vida que tem
         */
        private void mudarTransparencia()
        {
            if (isTouching(Gas.class))
            {
                vida--;
                if(vida<=0) //este if evita que a transparencia seja um valor negativo, o que gera erro de compilação
                {
                    getWorld().removeObject(this);
                }
                else
                {
                    imageCamada.setTransparency(2*vida+5);
                    removeTouching(Gas.class);
                }

            }
        }

    }
```

6.2.3.1. VidaCamadaOzono

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class VidaCamadaOzono extends CamadaOzono
{
  private int health = CamadaOzono.getVida();
  private final int HEALTHBARCOMPRIMENTO = 250;
  private final int HEALTHBARALTURA = 10;
  private final int percentagemDeVida = (int)HEALTHBARCOMPRIMENTO/health;

  public VidaCamadaOzono(){
     atualiza();
   }
```

```
    public void act()
    {
        atualiza();
    }

    /**
     * Método que atualiza a healthbar que mostra ao jogador quanta vida a camada de ozono tem
     */
    private void atualiza(){
        setImage(new GreenfootImage(HEALTHBARCOMPRIMENTO + 2, HEALTHBARALTURA + 2));
        GreenfootImage myImage = getImage();
        health=CamadaOzono.getVida();
        myImage.setColor(Color.BLACK);
        myImage.drawRect(0,0,HEALTHBARCOMPRIMENTO +1, HEALTHBARALTURA +1);
        myImage.setColor(Color.CYAN);
        myImage.fillRect(1,1,health*percentagemDeVida, HEALTHBARALTURA);
    }
}
```

6.2.4. GameOver

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class GameOver here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class GameOver extends Jogos
{
    private GreenfootImage pisca1, pisca2;
    private int contador;

    public GameOver(){
```

```
        pisca1 = new GreenfootImage("GM1.png");
        pisca2 = new GreenfootImage("gm2.png");
        setImage(pisca1);
        contador = 0;
    }

    public void act()
    {
        pisca();
    }


    private void pisca(){
        contador++;
        if (contador==10)
        {
            if(getImage()==pisca1)
            {
                setImage(pisca2);
            }
            else if (getImage()==pisca2)
            {
                setImage(pisca1);
            }
            contador=0;
        }
    }
}
```

6.2.5. Gas

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Gas extends Jogos
{

    public void act()
    {
        movimentoGas();
```

```
        ultrapassaLimite();
    }

    /**
     * Método que remove o objeto quando chega ao limite do mundo
     */
    private void ultrapassaLimite()
    {
        if(isAtEdge()){
            getWorld().removeObject(this);
        }
    }

    /**
     * Método que move o objeto para cima e talvez para os lados, para torná-lo mais realista
     */
    private void movimentoGas(){
        if (Greenfoot.getRandomNumber(100)<10){
            setLocation(getX()+ Greenfoot.getRandomNumber(11)-5, getY());
        }
        setLocation(getX(), getY() - 1);
    }
}
```

6.2.6. Mar

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Mar extends Jogos
{
    public void act()
    {
        move(-1);
        remove();
    }

    /**
     * Método que remove o objeto quando chega ao limite do mundo
     */
```

```
    private void remove()
    {
        if (isAtEdge()){
            getWorld().removeObject(this);
    }
}
}
```

6.2.7. Missil

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Missil extends Jogos
{
    private int limite;
    private Nave2 nave2;
    private Nave1 nave1;
    private boolean Nave2EAlvo;
    private int contador;
    private GreenfootImage[] animaExplosao;
    private int indice;
    private boolean explosao;

    /**
     * Construtor que define qual o alvo do missil e este é "pintado" de acordo com a cor do
alvo e
     * define a altura limite depois da qual o missil já não é teleguiado
     */
    public Missil(int valor,boolean alvoENave2, Nave1 N1, Nave2 N2)
    {
        limite=valor;
        nave2=N2;
        nave1=N1;
        explosao=false;
        if (alvoENave2) //o missil fica com a cor do alvo
        {
            setImage(Player2.getColor()+"/Missil.png");
        }
        else
```

```
    {
        setImage(Player1.getColor()+"/Missil.png");
    }
    Nave2EAlvo=alvoENave2;
    animaExplosao = new GreenfootImage[15];
    getImage().scale(getImage().getWidth()/8, getImage().getHeight()/8);
    for (int i=0; i < animaExplosao.length; i++)
    {
        animaExplosao[i] = new GreenfootImage("ExplosionMisseis/"+(i+1)+".png");
    }
    contador=0;
    indice=0;
}

public void act()
{
    geral();
}

/**
 * Método responsável por fazer as ações do missil
 */
private void geral()
{
    if(!explosao)
    {
        if (Nave2EAlvo)
        {
            teleguiado(nave2);
        }
        else
        {
            teleguiado(nave1);
        }
        atingiuAlvo();
        if(!explosao){
            chegouAoFim();
```

```
            }
        }
        else
        {
            animaExplosao();
        }
    }

    /**
     * Métodos que vão teleguiar o míssil consoante a posição do alvo
     */
    private void teleguiado(Nave2 alvo){
        if(getY()>limite)
        {
            if (!getWorld().getObjects(Nave2.class).isEmpty()) //se a nave for destruida antes deste missil sair do mundo, este if evita que se va buscar a localização
            {
                turnTowards(alvo.getX(), alvo.getY());
            }
        }
        move(2);
    }

    private void teleguiado(Nave1 alvo){
        if(getY()>limite)
        {
            if (!getWorld().getObjects(Nave1.class).isEmpty()) //se a nave for destruida antes deste missil sair do mundo, este if evita que se va buscar a localização
            {
                turnTowards(alvo.getX(), alvo.getY());
            }
        }
        move(2);
    }

    /**
     * Método que verifica que este chegou a um limite do mundo
```

```
     */
    private void chegouAoFim(){
       if(isAtEdge())
       {
          explosao=false;
          getWorld().removeObject(this);
       }
    }

    /**
     * Métodos que registam que o míssil atingiu uma das naves, retirando ao jogador cuja nave foi atingida vida e pontuação
     */
    private void atingiuAlvo()
    {
       if (isTouching(Nave2.class))
       {
          Player2.adicionaNumeroVidas(-1);
          Player2.adicionaScore(-100);
          setLocation(getX(), getY()-getImage().getHeight()); // poe a explosao do missil mais para cima, para parecer que é na nave
          explosao=true;
          getImage().clear();
       }
       else if (isTouching(Nave1.class))
       {
          Player1.adicionaNumeroVidas(-1);
          Player1.adicionaScore(-100);
          setLocation(getX(), getY()-getImage().getHeight());// poe a explosao do missil mais para cima, para parecer que é na nave
          explosao=true;
          getImage().clear();
       }
    }

    /**
     * Método que anima a explosão do míssil quando este atinge uma nave
```

```
     */
    private void animaExplosao()
    {
        if (contador%3==0)
        {
            if(indice==0)
            {
                GreenfootSound som = new GreenfootSound("explosion.mp3");
                som.play();
                som.setVolume(15);
            }
            setImage(animaExplosao[indice]);
            indice++;
            if(indice>=animaExplosao.length)
            {
                getWorld().removeObject(this);
            }
        }
        contador++;
    }
}
```

6.2.8. Máquina

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Máquina extends Jogos
{
    private static int vida;
    private static int contador;
    private GreenfootImage[] explosao;
    private int indice;

    public Máquina(){
        vida=500;
        explosao = new GreenfootImage[13];
        for (int i=0; i< explosao.length; i++)
        {
            explosao[i] = new GreenfootImage("ExplosionMáquina/"+(i+1)+".png");
```

```
        }
        contador=0;
        indice=0;
    }

    public void act()
    {
        libertaGas();
        maquinaDestruida();
    }

    public static int getVida()
    {
        return vida;
    }

    public static void tiraVida(int valor)
    {
        vida-=valor;
    }

    /**
     * Método que invoca objetos da classe Gas, de modo a mostrar que a máquina está a poluir a atmosfera
     */
    private void libertaGas(){
        if (Greenfoot.getRandomNumber(100)>97 && vida > 0){
            getWorld().addObject(new     Gas(),this.getX()+Greenfoot.getRandomNumber(40)-10, this.getY() - this.getImage().getHeight()/2);
        }
    }

    /**
     * Método que anima a explosão que acontece quando a máquina é destruída, toca o som da explosão e avança o mundo para mostrar o score
     */
    private void maquinaDestruida(){
```

```
        if (vida <= 0 && contador%3==0){
            Jogo1.getSomChuva().stop();
            if(indice==0)
            {
                Greenfoot.playSound("explosion.mp3");
            }
            setImage(explosao[indice]);
            indice++;
            if(indice>=explosao.length)
            {
                Greenfoot.setWorld(new Stage1Complete());
                getWorld().removeObject(this);
            }
        }
        contador++;
    }
}
```

6.2.8.1. VidaMáquina

```
import greenfoot.*;   //(World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class VidaMáquina extends Máquina
{
  private int health = Máquina.getVida();
  private final int HEALTHBARCOMPRIMENTO = 1000;
  private final int HEALTHBARALTURA = 15;
  private final int percentagemDeVida = (int)HEALTHBARCOMPRIMENTO/health;

  public void act()
   {
      atualiza();
   }

  /**
   * Método que atualiza a barra de vida que informa os jogadores quanta vida a máquina tem
   */
  public void atualiza(){
```

```
            setImage(new GreenfootImage(HEALTHBARCOMPRIMENTO + 2,
HEALTHBARALTURA + 2));
            GreenfootImage myImage = getImage();
            health=Máquina.getVida();
            myImage.setColor(Color.WHITE);
            myImage.drawRect(0,0,HEALTHBARCOMPRIMENTO +1,
HEALTHBARALTURA +1);
            myImage.setColor(new Color(109, 16, 120));
            myImage.fillRect(1,1,health * percentagemDeVida, HEALTHBARALTURA);
        }
    }
```

6.2.9. Nuvem

```
    import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

    public class Nuvem extends Jogos
    {
        private int aux;
        private int sentido;

        public Nuvem(int sentidoMovimento)
        {
            aux=0;
            sentido=sentidoMovimento;

        }

        public void act()
        {
            movimentoNuvem(sentido);
        }

        /**
         * Método que movimenta a nuvem
         */
        private void movimentoNuvem(int sentido){
            if (aux<105)
            {
```

```
            move(sentido*3);

        }
        if (Greenfoot.getRandomNumber(100)<2 && aux>=105){
            setLocation(getX()+ Greenfoot.getRandomNumber(20)-10, getY());
        }
        aux++;

    }

}
```

6.2.10. ObjetosCaem

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class ObjetosCaem extends Jogos
{
    public void act()
    {
    }

    /**
     * Método que remove o objeto quando toca no "Chão"
     */
    protected void ultrapassaLimite(){
        if(isTouching(Chão.class)){
            getWorld().removeObject(this);
        }
    }

    /**
     * Método que movimenta todos os objetos que "caem"
     */
    protected  void movimento()
    {
        setLocation(getX()-1, getY() + 8);
    }
}
```

6.2.10.1. Gota

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Gota extends ObjetosCaem
{
   public void act()
   {
      movimento();
      ultrapassaLimite();
   }
}
```

6.2.10.2. Granizo

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Granizo extends ObjetosCaem
{
   public void act()
   {
      movimento();
      ultrapassaLimite();
   }
}
```

6.2.10.3. Neve

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Neve extends ObjetosCaem
{
   public void act()
   {
      movimento();
      ultrapassaLimite();
   }
}
```

6.2.10.4. Vida

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Vida extends ObjetosCaem
```

```
{
    public void act()
    {
        movimentoVida();
        ultrapassaLimite();
    }

    /**
     * A "Vida" tem um movimento descendente mais lento que os restantes "ObjetosCaem",
para o jogo 1
     */
    private void movimentoVida()
    {
        setLocation(getX(), getY() + 4);
    }
}
```

6.2.10.4.1. Vida_jogo2

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Vida_jogo2 extends Vida
{
    public void act()
    {
        movimentoVida();
        ultrapassaLimite();
    }

    /**
     * "Overriding" do método movimentoVida() herdado da classe "Vida"
     * Método que define o movimento (ascendente) da vida, para o jogo 2
     */
    public void movimentoVida()
    {
        setLocation(getX(), getY() - 3);
    }
```

```
    /**
     * "Overriding" do método ultrapassaLimite() herdado da classe "ObjetosCaem"
     * Método que remove o objeto quando toca nos limites do mundo, pois este objeto não tocará num objeto do tipo "Chão" mas tocará
     * sim nos limites do mundo
     */
    protected void ultrapassaLimite()
    {
        if(isAtEdge()){
            getWorld().removeObject(this);
        }
    }
}
```

6.2.11. Pinguim

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Pinguim extends Jogos
{
    private int contador;
    private GreenfootImage[] animacao;
    private boolean caindo;
    private int indice;
    private GreenfootSound salvo;
    public Pinguim(){
        animacao = new GreenfootImage[4];
        indice=0;
        animacao[0] = new GreenfootImage("pinguim1.png");
        animacao[1] = new GreenfootImage("pinguim2.png");
        animacao[2] = new GreenfootImage("pinguim3.png");
        animacao[3] = new GreenfootImage("pinguimCaindo.png");
        setImage(animacao[indice]);
        contador=0;
        GreenfootSound som = new GreenfootSound("penguin.mp3");
        som.play();
        som.setVolume(20);
        this.salvo= new GreenfootSound("getsHealth.mp3");
        caindo = false;
```

```
    }

    public void act()
    {
       move(-1);
       animar();
       emCimaPlataforma();
       desaparece();
       caindo();
       salvado();
    }

    /**
     * Método para animar o pinguim quando este está em cima de uma plataforma de gelo, acenando aos jogadores
     */
    public void animar(){
       if(!caindo){
          contador++;
          if (contador%8==0){
             indice++;
             if (indice>animacao.length-2)
             {
                indice=1;
             }
             setImage(animacao[indice]);
             contador=0;
          }
       }
    }

    /**
     * Método que regista que pinguim foi salvo pelos esquimós
     */
    private void salvado()
    {
       if (isTouching(Esquimó1.class) && isTouching(Esquimó2.class))
```

```
        {
            salvo.play();
            salvo.setVolume(30);
            Player1.adicionaScore(25);
            Player2.adicionaScore(25);
            Jogo3.incrementarPinguimSalvo();
            getWorld().removeObject(this);
        }
    }

    /**
     * Método para alterar a posição do objeto (para parecer que está a cair) quando deixa de estar em contacto com uma plataforma de gelo
     * Altera também a variável "caindo", para depois ser utilizada no método caindo() e alterar a animação do objeto
     */

    public void emCimaPlataforma(){
        if (!isTouching(PlataformaGelo.class))
        {
            setLocation(getX(), getY() +4);
            caindo=true;
        }
        else if (isTouching(PlataformaGelo.class) && caindo)
        {
            caindo=false;
        }
    }


    /**
     * Método para alterar a animação do objeto quando está a "cair" (quando a variável caindo é igual a true)
     */
    public void caindo(){
        if(caindo){
            indice=3;
```

```
            setImage(animacao[indice]);
            turn(10);
        }
    }

    /**
     * Método para remover o objeto quando toca no limite inferior do mundo
     */
    public void desaparece(){
        if (getY() == getWorld().getHeight()-1){
            Player1.adicionaScore(-200);
            Player2.adicionaScore(-200);
            getWorld().removeObject(this);
        }
    }
}
```

6.2.12. PlataformaGelo

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class PlataformaGelo extends Jogos
{
    private boolean pinguim;
    private GreenfootImage imagemAtual;
    public PlataformaGelo(int subtrair){
        podeCriarPinguim();
        imagemAtual = new GreenfootImage("platf1.png");
        //A largura das plataformas será tanto maior quanto menor foi o dano à camada de ozono
no jogo 2
        imagemAtual.scale(imagemAtual.getWidth()-(125-
CamadaOzono.getVida())/5+subtrair, imagemAtual.getHeight());
        setImage(imagemAtual);
    }

    public PlataformaGelo(){
    }

    public void act()
```

```
        {
            move(-1);
            derreter();
            desaparecer();
            invocaPinguim();
        }

        /**
         * Método para simular que as plataformas de Gelo estão a derreter, para alterar a tranparência do objeto quando estiver quase alcançando o
         * limite esquerdo do mundo, e fazer com que a plataforma pareça que está a cair quando aproxima o limite esquerdo do mundo
         */
        public void derreter(){
            if (Greenfoot.getRandomNumber(100)<=1){
                getWorld().addObject(new Gota(), getX() + Greenfoot.getRandomNumber(50)-50, getY());
            }
            if(getX()%75==0)
            {
                imagemAtual.setTransparency(imagemAtual.getTransparency()-10);
            }
            else if(getX()<40)
            {
                imagemAtual.setTransparency(70);
                turn(-2);
                setLocation(getX(), getY()+8);
            }
            setImage(imagemAtual);
        }


        public void podeCriarPinguim(){
            if (Greenfoot.getRandomNumber(100)<=20){
                pinguim=true;
            }else{
                pinguim=false;
```

```
            }
        }

        /**
         * Método para adicionar objetos da classe "Pinguim" em cima das plataformas
         */
        public void invocaPinguim(){
            if (pinguim == true){
                Pinguim peng = new Pinguim();
                getWorld().addObject(peng,                    getX()                    -
Greenfoot.getRandomNumber(getImage().getWidth()/2),    getY()-getImage().getHeight()/2    -
peng.getImage().getHeight()/2);
                pinguim = false;
                peng=null;
            }
        }

        public void desaparecer(){
            if (getX() == 0 || getY() == getWorld().getHeight()-1){
                GreenfootSound som = new GreenfootSound("splash.mp3");
                som.play();
                som.setVolume(30);
                getWorld().removeObject(this);
            }
        }
    }
```

6.2.12.1. Plataforma_Final

```
    import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

    public class Plataforma_Final extends PlataformaGelo
    {
        private GreenfootImage image;
        private int contador;

        public Plataforma_Final()
        {
            contador =0;
```

```
        setImage("plat_final.png");
    }

    public void act()
    {
        permanecerParado();
    }

    public void permanecerParado()
    {
        if (contador<100)
        {
            move(-1);
        }
        contador++;

    }
}
```

6.2.12.2. Plataforma_Inicial

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Plataforma_Inicial extends PlataformaGelo
{
    private int contador;
    private GreenfootImage imagemAtual;

    public Plataforma_Inicial()
    {
        imagemAtual = new GreenfootImage("platf1.png");
        //A largura das plataformas será tanto maior quanto menor foi o dano à camada de ozono
no jogo 2
        imagemAtual.scale(imagemAtual.getWidth()-(125-CamadaOzono.getVida())/5,
imagemAtual.getHeight());
        setImage(imagemAtual);
        contador =0;
    }
```

```
    public void act()
    {
        desaparecer();
        permanecerParado();
    }

    public void permanecerParado()
    {
        if (contador >650)
        {
            move(-1);
        }
        contador++;

    }
}
```

6.2.13. Relâmpago

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Relâmpago here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Relâmpago extends Jogos
{
    private GreenfootImage[] relampago;
    private int indice;
    private int count;
    private final int TIMER;
    private GreenfootSound somRelampago;
    public Relâmpago()
    {
        somRelampago = new GreenfootSound ("relampago.mp3");
        relampago= new GreenfootImage[3];
        for (int i=0; i < relampago.length; i++)
```

```
    {
        relampago[i] = new GreenfootImage("Lightning/"+(i+1)+".png");
        relampago[i].scale(3*relampago[i].getWidth()/4,relampago[i].getHeight());
    }
    setImage(relampago[indice]);
    indice=0;
    indice++;
    somRelampago.play();
    somRelampago.setVolume(35);
    TIMER=8;
    count = 0;
}

public void act()
{
    switchImage();
}

/**
 * Método que anima o relâmpago
 */
public void switchImage()
{
    if (count%TIMER==0){
        if (count < TIMER*relampago.length)
        {
            setImage(relampago[indice]);
            indice++;
            if (indice >= relampago.length)
            {
                indice=0;
            }
        }
        else
        {
            getWorld().removeObject(this);
        }
```

```
        }
        count++;
    }
}
```

6.2.14. Target

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class target here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Target extends Jogos
{
    private final int TIMER;
    private int count2;
    private GreenfootImage semBrilho,comBrilho;

    private int contador;
    public Target(){
        semBrilho = new GreenfootImage("targetf.png");
        comBrilho = new GreenfootImage("target8.png");
        setImage(semBrilho);
        TIMER=5;
        //count = TIMER*4;
        count2 = 0;
        contador=0;
    }

    public void act()
    {
        switchImage();
        conta(this);
    }

    public void switchImage()
```

```
        {
            if (count2%TIMER==0){
                if (getImage() == semBrilho)
                {
                    setImage(comBrilho);
                }

                else if (getImage() == comBrilho)
                {
                    setImage(semBrilho);
                }
            }
            count2++;
        }

        /**
         * Método que faz a contagem para remover o objeto do mundo e adicionar um novo objeto da classe "Relâmpago"
         * (permite "dar tempo" aos jogadores para se afastarem do relâmpago)
         */
        private void conta(Target target)
        {
            contador++;
            if(contador == 140)
            {
              getWorld().addObject(new Relâmpago(), target.getX(),320);
              getWorld().removeObject(target);
              contador = 0;
            }
        }
    }
```

6.3. Classes *Actor*, "Menus" e subclasses

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Buttons here.
 *
```

```
 * @author (your name)
 * @version (a version number or a date)
 */
public class Menus extends Objetos
{
    private GreenfootSound click;
    public Menus()
    {
        click = new GreenfootSound("click.mp3");
    }
    public void act()
    {
    }
    /**
     * Método para alterar a imagem consoante o rato passa por cima do botão
     */
    protected void moveMouse(GreenfootImage image1, GreenfootImage image2){
        if(Greenfoot.mouseMoved(this)){
            setImage(image1);
        }
        if (Greenfoot.mouseMoved(null) && !Greenfoot.mouseMoved(this)){
            setImage(image2);
        }
    }

    /**
     * Método para tocar um som de "click" quando os botões são pressionados
     */
    protected void playClick()
    {
        click.play();
        click.setVolume(50);
    }
}
```

6.3.1. Back

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)
```

```
public class Back extends Menus
{
   private GreenfootImage image1, image2;
   public Back()
   {
      image1 = new GreenfootImage("BackSemBrilho.png");
      image2 = new GreenfootImage("BackComBrilho.png");
      setImage(image1);
   }

   public void act()
   {
      moveMouse(image2,image1);
      clickMouse();
   }

   /**
    * Métodos que registam que o rato clicou no botão ou o utilizador carregou no Esc e, neste caso, volta ao mundo anterior
    */
   private void clickMouse(){
      String key =Greenfoot.getKey();
      World mundo = getWorld();
      if (Greenfoot.mouseClicked(this) || (key!=null && key.equals("escape"))){
         playClick();
         Greenfoot.setWorld(new MenuInicial(false));
      }
      key = null;
      mundo = null;
   }
}
```

6.3.2. Cor

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Cor extends Menus
{
   private static boolean P1escolheu;
```

```
private String cor;

public Cor(int valor)
{
   GreenfootImage image = new GreenfootImage(150,150);
   switch (valor)
   {
      case 0:
      image.setColor(new Color(149,19,23)); //Vermelho
      cor = "Red";
      break;
      case 1:
      image.setColor(new Color(55,55,57)); //Preto
      cor = "Black";
      break;
      case 2:
      image.setColor(new Color(54,182,73)); //Verde
      cor = "Green";
      break;
      case 3:
      image.setColor(new Color(245,239,44)); //Amarelo
      cor = "Yellow";
      break;
      case 4:
      image.setColor(new Color(6,118,189)); //Azul
      cor = "Blue";
      break;
   }
   image.fillOval(0, 0, 150, 150);
   this.setImage(image);
   P1escolheu=false;
}

public void act()
{
   clickMouse();
}
```

```
        /**
         * Método que regista a cor que os jogadores escolheram e atualiza as variáveis e o mundo de acordo
         */
        private void clickMouse(){
            if (Greenfoot.mouseClicked(this)){
                playClick();
                if(!P1escolheu){
                    Player1.setColor(cor);
                    P1escolheu = true;
                    Texto.updateText(Player2.getNome()+",\npick your colour", getWorld().getObjects(Texto.class).get(0), 60, new Color(255,255,255));
                    getWorld().removeObject(this);
                }
                else{
                    Player2.setColor(cor);
                    Greenfoot.setWorld(new HowToPlay());
                }
            }
        }
    }
```

6.3.3. Exit

```
    import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

    /**
     * Write a description of class Exit here.
     *
     * @author (your name)
     * @version (a version number or a date)
     */
    public class Exit extends Menus
    {
        private GreenfootImage image1, image2;
        public Exit()
        {
            image1 = new GreenfootImage("EXITsem.png");
```

```
        image2 = new GreenfootImage("EXITcom.png");
    }

    public void act()
    {
        moveMouse(image2,image1);
        clickMouse();
    }

    private void clickMouse(){
        String key = Greenfoot.getKey();
        if (Greenfoot.mouseClicked(this) || (key!=null && key.equals("escape"))){
            playClick();
            MenuInicial.getMusica().stop();
            Greenfoot.stop();
        }
        key=null;
    }
}
```

6.3.4. Options

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Options here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Options extends Menus
{
    private GreenfootImage image1, image2;
    public Options()
    {
        image1 = new GreenfootImage("OptionsSemBrilho.png");
        image2 = new GreenfootImage("OptionsComBrilho.png");
    }
```

```
    public void act()
    {
        moveMouse(image2,image1);
        clickMouse();
    }

    public void clickMouse(){
        if (Greenfoot.mouseClicked(this)){
            playClick();
            Greenfoot.setWorld(new Opções());
        }
    }
}
```

6.3.4.1. Controlos

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Controlos here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Controlos extends Options
{
    private Boolean flag;
    private Boolean prepara;
    private int player;
    private int controlo;
    private Texto text;
    private GreenfootImage image1, image2;
    public Controlos (int player, int controlo)
    {
        this.flag = true;
        this.prepara=true;
        this.player=player;
        this.controlo=controlo;
        this.text = new Texto();
```

```
        this.image1 = new GreenfootImage("BotaoControlos.png");
        this.image2 = new GreenfootImage("BotaoControlos2.png");
        setImage(image1);
    }

    public void act()
    {
        inicio();
        moveMouse(image2,image1);
        clickMouse(player,controlo);
    }

    /**
     * Método que insere o texto que demonstra qual o controlo atual.
     */
    private void inicio(){
        if (prepara){
            getWorld().addObject(text, getX(), getY());
            if (player == 1)
            {
                text.updateText(Player1.getControls()[controlo],text,40,                    new
Color(255,255,255));
            }
            else
            {
                text.updateText(Player2.getControls()[controlo],text,40,                    new
Color(255,255,255));
            }
            prepara=false;
            image1.scale((int)(1.5*getWorld().getWidth()/9), getWorld().getHeight()/9);
            image2.scale((int)(1.5*getWorld().getWidth()/9), getWorld().getHeight()/9);
        }
    }

    /**
     * Método que regista que o utilizador clicou no botão para mudar o controlo escolhido e
espera que o mesmo dê uma tecla válida,
```

```
     * não pode escolher uma tecla que esteja ocupada para outro controlo, quer do player 1 quer do player 2
     */
    private void clickMouse(int player, int controlo){
        if (Greenfoot.mouseClicked(this) || Greenfoot.mouseClicked(text)){
            playClick();
            Greenfoot.getKey();
            text.updateText("Press any key",text,30, new Color(255,255,255));
            Greenfoot.delay(1);
            while (flag){
                String key= Greenfoot.getKey();
                if (key!=null && Players.podeMudar(Player1.getControls(),Player2.getControls(),key)){
                    if (player == 1)
                    {
                        Player1.setControls(controlo,key);
                    }
                    else
                    {
                        Player2.setControls(controlo,key);
                    }

                    flag=false;
                    if(!flag){
                        if (player == 1)
                        {
                            text.updateText(Player1.getControls()[controlo],text,40, new Color(255,255,255));
                        }
                        else
                        {
                            text.updateText(Player2.getControls()[controlo],text,40, new Color(255,255,255));
                        }
                    }
                }
            }
```

```
            flag=true;
        }
    }
}
```

6.3.4.2. Texto

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Texto here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Texto extends Options
{
    private Color Cor;

    public Texto(){}

    public Texto(String text, int size , Color cor){
        GreenfootImage image = new GreenfootImage(text.toUpperCase(),size, cor, new Color(0,0,0,0));
        setImage(image);
        Cor=cor;
    }

    public void act()
    {
    }

    public Color getCor()
    {
        return Cor;
    }
    /**
     * Método para atualizar o texto mostrado pela classe Texto
     */
```

```
    public static void updateText(String texto, Actor text, int size, Color cor){
        GreenfootImage image = new GreenfootImage(texto.toUpperCase(),size, cor, new
Color(0,0,0,0));
        text.setImage(image);
    }

}
```

6.3.5. Play

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class Play extends Menus
{
    private GreenfootImage image1, image2;
    public Play()
    {
        image1 = new GreenfootImage("Play1.png");
        image2 = new GreenfootImage("Play2.png");
    }

    public void act()
    {
        moveMouse(image2,image1);
        clickMouse();
    }

    private void clickMouse(){
        if (Greenfoot.mouseClicked(this)){
            playClick();
            Greenfoot.setWorld(new EscolhaNomes());
        }
    }

}
```

6.3.6. Restart

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
```

```
 * Write a description of class Restart here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Restart extends Menus
{
   private GreenfootImage image1, image2;
   public Restart()
   {
      image1 = new GreenfootImage("RESTART (1).png");
      image2 = new GreenfootImage("RESTART.png");
   }
   public void act()
   {
      moveMouse(image1,image2);
      clickMouse();
   }

   private void clickMouse(){
      if (Greenfoot.mouseClicked(this)){
         playClick();
         Greenfoot.setWorld(new MenuInicial(false));
      }
   }
}
```

6.3.7. Stage

6.3.7.1. GoBack

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class GoBack here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class GoBack extends Stage
```

```
{
   private GreenfootImage image1,image2;

   public GoBack(){
      image1 = new GreenfootImage("goBack1.png"); //sem brilho
      image2 = new GreenfootImage("goBack2.png"); //com brilho
      setImage(image1);
   }

   public void act()
   {
      moveMouse(image2,image1);
      clickMouse();
   }

   private void clickMouse(){
      if (Greenfoot.mouseClicked(this)){
         playClick();
         if(IndividualScore.getJogo1())
         {
            Greenfoot.setWorld(new Stage1Complete());
         }
         else
         {
            Greenfoot.setWorld(new Stage2Complete());
         }
      }
   }
}
```

6.3.7.2. Menu

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Menu here.
 *
 * @author (your name)
 * @version (a version number or a date)
```

```
 */
public class Menu extends Stage
{
   private GreenfootImage image1,image2;

   public Menu(){
      image1 = new GreenfootImage("close1.png"); //sem brilho
      image2 = new GreenfootImage("close.png"); //com brilho
      setImage(image1);
   }
   public void act()
   {
      moveMouse(image2,image1);
      clickMouse();
   }

   private void clickMouse(){
      if (Greenfoot.mouseClicked(this)){
         playClick();
         Greenfoot.setWorld(new MenuInicial(false));
      }
   }
}
```

6.3.7.3. NextLevel

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class NextLevel here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class NextLevel extends Stage
{
   private GreenfootImage image1,image2;

   public NextLevel(){
```

```
        image1 = new GreenfootImage("nextLevel1.png"); //sem brilho
        image2 = new GreenfootImage("nextLevel.png"); //com brilho
        setImage(image1);
    }
    public void act()
    {
        moveMouse(image2,image1);
        clickMouse();
    }

    private void clickMouse(){
        if (Greenfoot.mouseClicked(this)){
            playClick();
          Greenfoot.setWorld(new HowToPlay());
        }
    }
}
```

6.3.7.4. PlayersScore

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

public class PlayersScore extends Stage
{
    private GreenfootImage image1,image2;

    public PlayersScore(){
        image1 = new GreenfootImage("playersScore1.png"); //sem brilho
        image2 = new GreenfootImage("playersScore.png"); //com brilho
        setImage(image1);
    }
    public void act()
    {
        moveMouse(image2,image1);
        clickMouse();
    }

    private void clickMouse(){
        if (Greenfoot.mouseClicked(this)){
```

```
            playClick();
              Greenfoot.setWorld(new IndividualScore());
        }
    }
}
```

6.3.8. Start

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)
public class Start extends Stage
{
    private GreenfootImage image1,image2;
    public Start(){
        image1 = new GreenfootImage("StartSemBrilho.png");
        image2 = new GreenfootImage("StartComBrilho.png");
        setImage(image1);
    }

    public void act()
    {
        moveMouse(image2,image1);
        clickMouse();
    }

    private void clickMouse(){
        if (Greenfoot.mouseClicked(this)){
            playClick();
            MenuInicial.getMusica().stop();
            if (getWorld() instanceof HowToPlay){
                if(HowToPlay.jogoAtual() == 1){
                    Greenfoot.setWorld(new Jogo1());
                }else if(HowToPlay.jogoAtual() == 2){
                    Greenfoot.setWorld(new Jogo2());
                }else if(HowToPlay.jogoAtual() == 3){
                    Greenfoot.setWorld(new Jogo3());
                }
            }
        }
    }
```

}

6.4. Classes *Actor*, "Players" e subclasses

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Players here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Players extends Actor
{
   private boolean tocandoRelampago;
   private int contador;
   protected int indice;
   private int auxDeath;
   private int indiceDeath;
   private boolean control;
   protected static final int GRAVIDADE =15;
   protected int tempoJump;
   protected int tempoQueda;
   protected boolean podeSaltar;
   protected boolean saltou;
   protected boolean andandoParaEsquerda;
   protected int controlBala;
   private GreenfootSound atingido, apanhouVida;

   public Players()
   {
      tocandoRelampago = false;
      contador=0;
      indice=0;
      auxDeath=0;
      indiceDeath=0;
      control=true;
      tempoJump=GRAVIDADE;
```

```
        tempoQueda=GRAVIDADE;
        podeSaltar=false;
        saltou=false;
        andandoParaEsquerda=false;
        controlBala=0;
        atingido = new GreenfootSound("PlayerHit.mp3");
        apanhouVida = new GreenfootSound("getsHealth.mp3");
    }

    public void act()
    {
    }

    /**
     * Métodos que registam a perda, ou ganho, de vidas consoante o objeto que os jogadores tocam no primeiro jogo
     */
    protected void perdeVidas(Player1 P1)
    {
        if(P1.getNumeroVidas()>0){
            if(isTouching(Granizo.class))
            {
                playAtingido();
                P1.adicionaNumeroVidas(-1);
                P1.adicionaScore(-50);
                removeTouching(Granizo.class);
            }

            if(isTouching(Relâmpago.class))
            {
                if (!tocandoRelampago)
                {
                    P1.adicionaNumeroVidas(-2);
                    P1.adicionaScore(-100);
                    tocandoRelampago = true;
                }
            }
```

```
        else
        {
            tocandoRelampago = false;
        }

        if(isTouching(Vida.class) )
        {
            playVida();
            P1.adicionaNumeroVidas(2);
            P1.adicionaScore(25);
            removeTouching(Vida.class);

        }
    }

}

protected void perdeVidas(Player2 P2)
{
    if(P2.getNumeroVidas()>0){
        if(isTouching(Granizo.class))
        {
            playAtingido();
            P2.adicionaNumeroVidas(-1);
            P2.adicionaScore(-50);
            removeTouching(Granizo.class);
        }
        if(isTouching(Relâmpago.class))
        {
            if (!tocandoRelampago)
            {
                P2.adicionaNumeroVidas(-2);
                P2.adicionaScore(-100);
                tocandoRelampago = true;
            }
        }
        else
```

```
        {
            tocandoRelampago = false;
        }
        if(isTouching(Vida.class))
        {
            playVida();
            P2.adicionaNumeroVidas(2);
            P2.adicionaScore(25);
            removeTouching(Vida.class);
        }
    }
}

/**
 * Métodos que registam que as naves apanharam vida no segundo jogo
 */
protected void perdeVidas(Nave1 N1)
{
    if(N1.getNumeroVidas()>0)
    {
        if(isTouching(Vida_jogo2.class))
        {
            playVida();
            N1.adicionaNumeroVidas(2);
            N1.adicionaScore(25);
            removeTouching(Vida_jogo2.class);
        }
    }

}

protected void perdeVidas(Nave2 N2)
{
    if(N2.getNumeroVidas()>0)
    {
        if(isTouching(Vida_jogo2.class))
        {
```

```
            playVida();
            N2.adicionaNumeroVidas(2);
            N2.adicionaScore(25);
            removeTouching(Vida_jogo2.class);
        }

    }

}

/**
 * Método que trata da animação do movimento do jogador
 */
protected void animarMove(GreenfootImage[] animacao){
    contador++;
    if (contador==4){
        if(indice<animacao.length-1)
        {
            indice++;
        }
        else
        {
            indice=1;
        }
        setImage(animacao[indice]);
        contador=0;
    }
}

/**
 * Método que trata da animação da morte do jogador
 */
protected void animarMorte(GreenfootImage[] animacaoDeath, boolean
andandoParaEsquerda)
{
    if(andandoParaEsquerda && control)
    {
```

```
        for (int i=0; i < animacaoDeath.length;i++)
        {
            animacaoDeath[i].mirrorHorizontally();
        }
        control=false;
    }
    if(auxDeath%10==0)
    {
        Player1.setP2Morreu(true);
        Player2.setP1Morreu(true);
        setImage(animacaoDeath[indiceDeath]);
        indiceDeath++;
        if (indiceDeath>=animacaoDeath.length)
        {
            getWorld().removeObject(this);
        }
    }
    auxDeath++;
}

/**
 * Método que anima a explosão quando a nave é destruída
 */
protected void naveDestruida(GreenfootImage[] explosao){
    if (contador%3==0){
        Player1.setP2Morreu(true);
        Player2.setP1Morreu(true);
        if(indice==0)
        {
            Greenfoot.playSound("explosion.mp3");
        }
        setImage(explosao[indice]);
        indice++;
        if(indice>=explosao.length)
        {
            getWorld().removeObject(this);
        }
```

```
    }
    contador++;
}

/**
 * Métodos que registam que o esquimó caiu ao mar
 */
protected void cair(Esquimó1 morto, GreenfootImage[] animacao){
    if (isTouching(Mar.class)){
        morto.adicionaNumeroVidas(-10);
        Player2.setP1Morreu(true);
        Player1.setP2Morreu(true);
        animacao[indice].setTransparency(animacao[indice].getTransparency()-5);
    }

    if(animacao[indice].getTransparency() <=0){
        getWorld().removeObject(this);
    }
}

protected void cair(Esquimó2 morto, GreenfootImage[] animacao){
    if (isTouching(Mar.class)){
        morto.adicionaNumeroVidas(-10);
        Player2.setP1Morreu(true);
        Player1.setP2Morreu(true);
        animacao[indice].setTransparency(animacao[indice].getTransparency()-5);
    }

    if(animacao[indice].getTransparency() <=0){
        getWorld().removeObject(this);
    }
}

/**
 * Método que toca um som quando o jogador é atingido por uma bola de granizo
 */
public void playAtingido()
```

```
        {
            atingido.play();
            atingido.setVolume(50);
        }

        /**
         * Método que toca um som quando o jogador apanha um coração de vida
         */
        public void playVida()
        {
            apanhouVida.play();
            apanhouVida.setVolume(30);
        }

        /**
         * Método usado na mudança dos controlos, verifica se a tecla inserida não causa conflito com os outros controlos
         */
        public static boolean podeMudar(String[] P1Controls, String[] P2Controls, String key){
            for (int i=0; i<4;i++){
                if (key.equals(P1Controls[i]) || key.equals(P2Controls[i])){
                    return false;
                }
            }
            return true;
        }
    }
```

6.4.1. Bala

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Bala here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Bala extends Players
```

```
{
    private GreenfootImage original;
    private boolean moveEsquerda, mudaOrientacao;
    private final int VELOCIDADE;
    private Player1 P1Disparou;
    private Player2 P2Disparou;

    private GreenfootSound disparo;

    public Bala(Player1 P1){
        disparo = new GreenfootSound("gun.mp3");
        original = getImage();
        setImage(original);
        moveEsquerda= P1.getAndandoParaEsquerda();
        disparo.play();
        disparo.setVolume(15);
        VELOCIDADE=8;
        P1Disparou = P1;
        mudaOrientacao = false;
    }

    public Bala(Player2 P2){
        disparo = new GreenfootSound("gun.mp3");
        original = getImage();
        setImage(original);
        moveEsquerda= P2.getAndandoParaEsquerda();
        disparo.play();
        disparo.setVolume(15);
        VELOCIDADE=8;
        P2Disparou = P2;
        mudaOrientacao = false;
    }

    public void act()
    {
        disparo();
        if(desapareceLimite() ||atingiuMáquina())
```

```
    {
        getWorld().removeObject(this);
    }
}

/**
 * Método que movimenta a bala
 */
private void disparo(){
    if(!mudaOrientacao)
    {
        if (moveEsquerda){
            original = getImage();
            original.mirrorHorizontally();
            setImage(original);
        }
        mudaOrientacao = true;
    }
    else{
        if (moveEsquerda){
            move(-VELOCIDADE);
        }
        else{
            move(VELOCIDADE);
        }
    }
}

private boolean desapareceLimite(){
    if (isAtEdge()){
        return true;
    }
    return false;
}

/**
```

```
     * Método que regista que a bala atingiu a máquina e adiciona score ao jogador que disparou
     */
    private boolean atingiuMáquina()
    {
      if (isTouching(Máquina.class) && getWorld().getObjects(Máquina.class).get(0).getVida()>0)
      {
        if (P1Disparou!=null)
        {
          P1Disparou.adicionaScore(10);
        }
        else
        {
          P2Disparou.adicionaScore(10);
        }
        getWorld().getObjects(Máquina.class).get(0).tiraVida(1);
        return true;
      }
      return false;
    }
  }
```

6.4.2. Player1

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Player1 here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Player1 extends Players
{
  protected static String up;
  protected static String left;
  protected static String right;
  protected static String shoot;
```

```
private static String[] controlos= {up,left,right,shoot};
private static String cor;
private static String nome;
private GreenfootImage[] animacao;
private GreenfootImage[] animacaoDeath;
private static int numeroVidas=10;
private static int score=0;
private static boolean P2morreu;

public Player1()
{
   //animação movimento jogador:
   animacao=new GreenfootImage[7];
   animacao[0] = new GreenfootImage(cor+"/Idle.png");
   for(int i=1; i <animacao.length;i++)
   {
      animacao[i]=new GreenfootImage(cor+"/"+i+".png");
   }
   setImage(animacao[indice]);
   //animação do jogador a "morrer"
   animacaoDeath = new GreenfootImage[8];
   for(int i=0; i <animacaoDeath.length;i++)
   {
      animacaoDeath[i]=new GreenfootImage(cor+"/Death/"+(i+1)+".png");
   }
   P2morreu=false;
}

public void act()
{
   queda();
   move();
   disparar();
   perdeVidas(this);
   if(numeroVidas <=0)
   {
      animarMorte(animacaoDeath,andandoParaEsquerda);
```

```
    }
}

public static boolean getP2Morreu()
{
    return P2morreu;
}

public static void setP2Morreu(boolean x)
{
    P2morreu = x;
}

public static String getNome()
{
    return nome;
}

public static void setNome(String texto)
{
    nome=texto;
}

public static int getScore()
{
    return score;
}

public static void resetScore()
{
    score=0;
}

public static void adicionaScore(int valor)
{
    score+=valor;
    if(score<0)
```

```
    {
        score-=valor;
    }
}

public boolean getAndandoParaEsquerda()
{
    return andandoParaEsquerda;
}

public static String[] getControls()
{
    return controlos;
}

public static void setControls(int pos, String key)
{
    controlos[pos]=key;
    switch(pos){
        case 0:
        up=key;
        break;
        case 1:
        left=key;
        break;
        case 2:
        right=key;
        break;
        case 3:
        shoot=key;
        break;
    }

}

public static void setColor(String color)
{
```

```
    cor=color;
}

public static String getColor(){
    return cor;
}

public static int getNumeroVidas()
{
    return numeroVidas;
}

public static void resetNumVidas()
{
    numeroVidas=10;
}

public static void adicionaNumeroVidas(int valor)
{
    if (valor<0 || (valor > 0 && numeroVidas<9))
    {
        numeroVidas += valor;
    }
    else
    {
        if(numeroVidas<10)
        {
            numeroVidas++;
        }
    }
}

/**
 * Método para o movimento do jogador
 */
protected void move()
{
```

```
        if(numeroVidas > 0 &&!P2morreu){
            if (Greenfoot.isKeyDown(up) && podeSaltar){ //certifica que so pode saltar se estiver no chao
                saltou=true;
                podeSaltar=false;
            }
            if(saltou){
                jump();
            }
            if (Greenfoot.isKeyDown(left)){
                if (!andandoParaEsquerda){//reflete as imagens,se necessário
                    for (int i=0; i < animacao.length;i++)
                    {
                        animacao[i].mirrorHorizontally();
                    }
                }
                setLocation(getX()-2, getY());
                andandoParaEsquerda=true;
                animarMove(animacao);
            }
            else if (Greenfoot.isKeyDown(right) && !isTouching(Máquina.class)){
                if (andandoParaEsquerda){//reflete as imagens,se necessário
                    for (int i=0; i < animacao.length;i++)
                    {
                        animacao[i].mirrorHorizontally();
                    }
                }
                setLocation(getX()+2, getY());
                andandoParaEsquerda=false;
                animarMove(animacao);
            }
            if (!Greenfoot.isKeyDown(right) && !Greenfoot.isKeyDown(up) && !Greenfoot.isKeyDown(left)){
                indice=0;
                setImage(animacao[indice]);
            }
        }
```

```
}

/**
 * Método que trata da parte da subida do salto
 */
protected void jump(){
   if (tempoJump>0){
      setLocation(getX(),getY()-tempoJump);
      tempoJump--;
   }
   else{
      tempoJump=GRAVIDADE;
      saltou=false;
   }
}

/**
 * Método que trata da parte da queda depois da parte da subida do salto ter sido efetuada
 */
protected void queda(){
   if (!isTouching(Chão.class) && !saltou){
      setLocation(getX(),getY()+tempoQueda);
      tempoQueda++;
   }
   else{
      tempoQueda=0;
      if(isTouching(Chão.class)){
         podeSaltar=true;
      }
   }
}

/**
 * Método que trata do disparo
 */
private void disparar(){
   controlBala++;
```

```
            if(numeroVidas > 0 &&!P2morreu){
               int sentido;
               if(andandoParaEsquerda)
               {
                  sentido=-1;
               }
               else
               {
                  sentido=1;
               }
               if (controlBala >15 && Greenfoot.isKeyDown(shoot)){
                  getWorld().addObject(new                                    Bala(this),
getX()+sentido*(2*getImage().getWidth()/3),getY());
                  controlBala = 0;
               }
            }
         }
      }
```

6.4.2.1. Esquimó1

```
      import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

      public class Esquimó1 extends Player1
      {
         private GreenfootImage[] animacao;

         public Esquimó1()
         {
            animacao=new GreenfootImage[5];
            for(int i=0; i <animacao.length;i++)
            {
               animacao[i]=new GreenfootImage(getColor()+"/Jogo3/"+(i+1)+".png");
            }
         }

         public void act()
         {
            queda();
```

```
            move();
            cair(this, animacao);
        }

        /**
         * Método que trata do movimento do esquimó
         */
        protected void move()
        {
            if(getNumeroVidas() > 0 && !getP2Morreu()){
                if (Greenfoot.isKeyDown(up) && podeSaltar){//certifica que so pode saltar se estiver na plataforma de gelo
                    saltou=true;
                    podeSaltar=false;
                }
                if(saltou){
                    jump();
                }
                if (Greenfoot.isKeyDown(left)){
                    if (!andandoParaEsquerda){//reflete as imagens,se necessário
                        for (int i=0; i < animacao.length;i++)
                        {
                            animacao[i].mirrorHorizontally();
                        }
                    }
                    setLocation(getX()-3, getY());
                    andandoParaEsquerda=true;
                    animarMove(animacao);
                }
                else if (Greenfoot.isKeyDown(right) ){
                    if (andandoParaEsquerda){//reflete as imagens,se necessário
                        for (int i=0; i < animacao.length;i++)
                        {
                            animacao[i].mirrorHorizontally();
                        }
                    }
                    setLocation(getX()+3, getY());
```

```
                andandoParaEsquerda=false;
                animarMove(animacao);
            }
            if (!Greenfoot.isKeyDown(right) && !Greenfoot.isKeyDown(up) &&
!Greenfoot.isKeyDown(left)){
                indice=0;
                setImage(animacao[indice]);
            }
        }
    }

    /**
     * Método que trata da parte da subida do salto
     */
    protected void jump(){
        if (tempoJump>0){
            setLocation(getX(),getY()-tempoJump);
            tempoJump--;
        }
        else{
            tempoJump=GRAVIDADE;
            saltou=false;
        }
    }

    /**
     * Método que trata da parte da descida do salto, evita que os jogadores fiquem a meio do
bloco e regista se chegou à plataforma final
     */
    protected void queda(){
        Actor plat = getOneObjectAtOffset(0,tempoQueda+getImage().getHeight()/2,
PlataformaGelo.class); //verifica se, após a queda, o jogador estara em cima ou "dentro" da
plataforma
        if (plat==null  && !saltou){
            setLocation(getX(),getY()+tempoQueda);
            tempoQueda++;
        }
```

```
            else if (plat!=null){
                setLocation(getX(),plat.getY()-                    plat.getImage().getHeight()/2-
getImage().getHeight()/2);//mete os jogadores mesmo em cima da plataforma
                tempoQueda=0;
                podeSaltar=true;
                if (plat instanceof Plataforma_Final)
                {
                   Jogo3.setP1Chegou(true);
                }
            }
            plat=null;
        }
    }
```

6.4.2.2. Nave1

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Nave1 here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Nave1 extends Player1
{
    private GreenfootImage[] explosao;

    public Nave1()
    {
        setImage(getColor()+"/Nave.png");
        getImage().scale(getImage().getWidth()/2,getImage().getHeight()/2);
        explosao = new GreenfootImage[13];
        for(int i=0; i <explosao.length;i++)
        {
            explosao[i]=new GreenfootImage("ExplosionMáquina/"+(i+1)+".png");
        }
    }
```

```
    public void act()
    {
       moveNave();
       removeGas();
       perdeVidas(this);
       if(getNumeroVidas()<=0)
       {
          naveDestruida(explosao);
       }
    }

    /**
     * Método que trata do movimento da nave
     */
    private void moveNave(){
       if (getNumeroVidas() > 0 && !getP2Morreu())
       {
          if (Greenfoot.isKeyDown(left)){
             setLocation(getX()-3, getY());
          }
          else if (Greenfoot.isKeyDown(right)){
             setLocation(getX()+3, getY());
          }
       }
    }

    /**
     * Método que trata de registar que a nave "apanhou" o gás, dando ao jogador 10 de pontuação
     */
    private void removeGas()
    {
       if (isTouching(Gas.class))
       {
          Player1.adicionaScore(10);
          removeTouching(Gas.class);
```

```
        }
    }

}
```

6.4.2.3. Vida_player1

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Vida_player1 here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Vida_player1 extends Player1
{
    private GreenfootImage[] vidas;

    public Vida_player1()
    {
        vidas = new GreenfootImage[10];
        for (int i=0; i < vidas.length; i++)
        {
            vidas[i] = new GreenfootImage("Vida/"+(i+1)+".png");
        }
    }

    public void act()
    {
        vidaPlayer1();
    }

    public void vidaPlayer1()
    {
        if (Player1.getNumeroVidas() >0)
        {
            setImage(vidas[Player1.getNumeroVidas()-1]);
        }
```

```
        else {
            getWorld().removeObject(this);
        }
    }
}
```

6.4.3. Player2

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Player2 here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Player2 extends Players
{
    protected static String up;
    protected static String left;
    protected static String right;
    protected static String shoot;
    private static String[] controlos= {up,left,right,shoot};
    private static String cor;
    private static String nome;
    private GreenfootImage[] animacao;
    private GreenfootImage[] animacaoDeath;
    private static int numeroVidas=10;
    private static int score=0;
    private static boolean P1morreu;

    public Player2()
    {
        //animação movimento jogador:
        animacao=new GreenfootImage[7];
        animacao[0] = new GreenfootImage(cor+"/Idle.png");
        for(int i=1; i <animacao.length;i++)
        {
            animacao[i]=new GreenfootImage(cor+"/"+i+".png");
```

```
    }
    setImage(animacao[indice]);
    //animação do jogador a "morrer"
    animacaoDeath = new GreenfootImage[8];
    for(int i=0; i <animacaoDeath.length;i++)
    {
        animacaoDeath[i]=new GreenfootImage(cor+"/Death/"+(i+1)+".png");
    }
    P1morreu=false;
}

public void act()
{
    queda();
    move();
    disparar();
    perdeVidas(this);
    if(numeroVidas<=0)
    {
        animarMorte(animacaoDeath,andandoParaEsquerda);
    }
}

public static boolean getP1Morreu()
{
    return P1morreu;
}

public static void setP1Morreu(boolean x)
{
    P1morreu = x;
}

public boolean getAndandoParaEsquerda()
{
    return andandoParaEsquerda;
}
```

```
public static String getNome()
{
   return nome;
}

public static void setNome(String texto)
{
   nome=texto;
}

public static int getScore()
{
   return score;
}

public static void resetScore()
{
   score=0;
}

public static void adicionaScore(int valor)
{
   score+=valor;
   if(score<0)
   {
      score-=valor;
   }
}

public static String[] getControls(){
   return controlos;
}

public static void setControls(int pos, String key){
   controlos[pos]=key;
   switch(pos){
```

```
        case 0:
        up=key;
        break;
        case 1:
        left=key;
        break;
        case 2:
        right=key;
        break;
        case 3:
        shoot=key;
        break;
    }

}

public static void setColor(String color){
    cor=color;
}

public static String getColor(){
    return cor;
}

public static int getNumeroVidas()
{
    return numeroVidas;
}

public static void resetNumVidas()
{
    numeroVidas=10;
}

public static void adicionaNumeroVidas(int valor)
{
    if (valor<0 || (valor > 0 && numeroVidas<9))
```

```
        {
            numeroVidas += valor;
        }
        else
        {
            if(numeroVidas<10)
            {
                numeroVidas++;
            }
        }
    }

    /**
     * Método para o movimento do jogador
     */
    protected void move()
    {
        if(numeroVidas > 0 && !P1morreu){
            if (Greenfoot.isKeyDown(up) && podeSaltar){//certifica que so pode saltar se estiver no chao
                saltou=true;
                podeSaltar=false;
            }
            if(saltou){
                jump();
            }
            if (Greenfoot.isKeyDown(left) && !isTouching(Máquina.class)){
                if (!andandoParaEsquerda){//reflete as imagens,se necessário
                    for (int i=0; i < animacao.length;i++)
                    {
                        animacao[i].mirrorHorizontally();
                    }
                }
                setLocation(getX()-2, getY());
                andandoParaEsquerda=true;
                animarMove(animacao);
            }
```

```
            else if (Greenfoot.isKeyDown(right)){
                if (andandoParaEsquerda){//reflete as imagens,se necessário
                    for (int i=0; i < animacao.length;i++)
                    {
                        animacao[i].mirrorHorizontally();
                    }
                }
                setLocation(getX()+2, getY());
                andandoParaEsquerda=false;
                animarMove(animacao);
            }
            if (!Greenfoot.isKeyDown(right) && !Greenfoot.isKeyDown(up) &&
!Greenfoot.isKeyDown(left)){
                indice=0;
                setImage(animacao[indice]);
            }
        }
    }

    /**
     * Método que trata da parte da subida do salto
     */
    protected void jump(){
        if (tempoJump>0){
            setLocation(getX(),getY()-tempoJump);
            tempoJump--;
        }
        else{
            tempoJump=GRAVIDADE;
            saltou=false;
        }
    }

    /**
     * Método que trata da parte da queda depois da parte da subida do salto ter sido efetuada
     */
    protected void queda(){
```

```
            if (!isTouching(Chão.class) &&!saltou){
                setLocation(getX(),getY()+tempoQueda);
                tempoQueda++;
            }
            else{
                tempoQueda=0;
                if(isTouching(Chão.class)){
                    podeSaltar=true;
                }
            }
        }

        /**
         * Método que trata do disparo
         */
        public void disparar(){
            controlBala++;
            if(numeroVidas > 0 && !P1morreu){
                int sentido;
                if(andandoParaEsquerda)
                {
                    sentido=-1;
                }
                else
                {
                    sentido=1;
                }
                if (controlBala >15 && Greenfoot.isKeyDown(shoot)){
                    getWorld().addObject(new                                    Bala(this),
getX()+sentido*(2*getImage().getWidth()/3),getY());
                    controlBala = 0;
                }
            }
        }
    }
```

6.4.3.1. Esquimó2

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)
```

```
public class Esquimó2 extends Player2
{
   private GreenfootImage[] animacao;

   public Esquimó2()
   {
      animacao=new GreenfootImage[5];
      for(int i=0; i <animacao.length;i++)
      {
         animacao[i]=new GreenfootImage(getColor()+"/Jogo3/"+(i+1)+".png");
      }
   }

   public void act()
   {
      queda();
      move();
      cair(this, animacao);
   }

   /**
    * Método que trata do movimento do esquimó
    */
   protected void move()
   {
      if(getNumeroVidas() > 0 && !getP1Morreu()){
         if (Greenfoot.isKeyDown(up) && podeSaltar){//certifica que so pode saltar se
estiver na plataforma de gelo
            saltou=true;
            podeSaltar=false;
         }
         if(saltou){
            jump();
         }
         if (Greenfoot.isKeyDown(left)){
            if (!andandoParaEsquerda){//reflete as imagens,se necessário
```

```
                for (int i=0; i < animacao.length;i++)
                {
                    animacao[i].mirrorHorizontally();
                }
            }
            setLocation(getX()-3, getY());
            andandoParaEsquerda=true;
            animarMove(animacao);
        }
        else if (Greenfoot.isKeyDown(right)){
            if (andandoParaEsquerda){//reflete as imagens,se necessário
                for (int i=0; i < animacao.length;i++)
                {
                    animacao[i].mirrorHorizontally();
                }
            }
            setLocation(getX()+3, getY());
            andandoParaEsquerda=false;
            animarMove(animacao);
        }
        if (!Greenfoot.isKeyDown(right) && !Greenfoot.isKeyDown(up) &&
!Greenfoot.isKeyDown(left)){
            indice=0;
            setImage(animacao[indice]);
        }
    }
}

/**
 * Método que trata da parte da subida do salto
 */
protected void jump(){
    if (tempoJump>0){
        setLocation(getX(),getY()-tempoJump);
        tempoJump--;
    }
    else{
```

```
            tempoJump=GRAVIDADE;
            saltou=false;
        }
    }

    /**
     * Método que trata da parte da descida do salto, evita que os jogadores fiquem a meio do bloco e regista se chegou à plataforma final
     */
    protected void queda(){
        Actor plat = getOneObjectAtOffset(0,tempoQueda+getImage().getHeight()/2, PlataformaGelo.class); //verifica se, após a queda, o jogador estara em cima ou "dentro" da plataforma
        if (plat==null   && !saltou){
            setLocation(getX(),getY()+tempoQueda);
            tempoQueda++;
        }
        else if (plat!=null ){
            setLocation(getX(),plat.getY()- plat.getImage().getHeight()/2- getImage().getHeight()/2);//mete os jogadores mesmo em cima da plataforma
            tempoQueda=0;
            podeSaltar=true;
            if (plat instanceof Plataforma_Final)
            {
                Jogo3.setP2Chegou(true);
            }
        }
        plat=null;
    }
}
```

6.4.3.2. Nave2

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Nave2 here.
 *
 * @author (your name)
```

```
 * @version (a version number or a date)
 */
public class Nave2 extends Player2
{
   private GreenfootImage[] explosao;

   public Nave2()
   {
      setImage(getColor()+"/Nave.png");
      getImage().scale(getImage().getWidth()/2,getImage().getHeight()/2);
      explosao = new GreenfootImage[13];
      for(int i=0; i <explosao.length;i++)
      {
         explosao[i]=new GreenfootImage("ExplosionMáquina/"+(i+1)+".png");
      }
   }

   public void act()
   {
      moveNave();
      removeGas();
      perdeVidas(this);
      if(getNumeroVidas()<=0)
      {
         naveDestruida(explosao);
      }
   }

   /**
    * Método que trata do movimento da nave
    */
   private void moveNave(){
      if (getNumeroVidas() > 0 && !getP1Morreu())
      {
         if (Greenfoot.isKeyDown(left)){
            setLocation(getX()-3, getY());
         }
```

```
            else if (Greenfoot.isKeyDown(right)){
                setLocation(getX()+3, getY());
            }
        }
    }

    /**
     * Método que trata de registar que a nave "apanhou" o gás, dando ao jogador 10 de pontuação
     */
    private void removeGas()
    {
        if (isTouching (Gas.class))
        {
            Player2.adicionaScore(10);
            removeTouching (Gas.class);

        }
    }
}
```

6.4.3.3. Vida_player2

```
import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)

/**
 * Write a description of class Vida_player2 here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Vida_player2 extends Player2
{
    private GreenfootImage[] vidas;

    public Vida_player2()
    {
        vidas = new GreenfootImage[10];
        for (int i=0; i < vidas.length; i++)
```

```
        {
            vidas[i] = new GreenfootImage("Vida/"+(i+1)+".png");
            vidas[i].mirrorHorizontally();
        }
    }

    public void act()
    {
        vidaPlayer2();
    }

    public void vidaPlayer2()
    {
        if (Player2.getNumeroVidas() >0)
        {
            setImage(vidas[Player2.getNumeroVidas()-1]);
        }
        else {
            getWorld().removeObject(this);
        }
    }

}
```