UNIVERSIDADE da MADEIRA

Relatório do Projeto 1 de POO
Barbie – The Game

Professores:

Luis Ferreira

Yuri Almeida

Sergi Bermúdez

Alunos:

Paulo Alexandre Rodrigues Alves n.º 2120722

Renato Gabriel Silva Pêssego n.º 2121922

João Tomás Correia Abreu nº 2118722

Funchal

26 Outubro 2023

Resumo

Neste primeiro projeto da cadeira de POO foi nos proposto a realização de um jogo com o tema baseado no filme “Barbie-The Movie”. Barbie e Ken encontram-se perdidos, necessitando de uma maneira de voltar para casa.Para tal, só existe uma forma de regressar a casa.Completando juntos, o desafio que os foi proposto.

Era uma tarde ensolarada na cidade de Brinquedópolis, mas num enlace tornou-se numa das tardes mais sombrias que Brinquedópolis viu.Foi nessa tarde,que Ken, foi raptado por robôs e submetido a experiências para o transformar num robô. Ken e Barbie receberam um convite para participar de um parkour mágico que os levaria através de diferentes mundos e desafios para, talvez, reverter o estado robótico do Ken.Sem hesitar, Barbie e Ken aceitaram a missão e depararam-se diante de vários obstáculos que os levariam a um enorme portal cintilante.

Logo perceberam que estavam no início de uma grande jornada. Uma voz misteriosa ecoou no ar, explicando as regras do desafio. "Barbie e Ken, bem-vindos ao Parkour dos Sonhos. Vocês devem trabalhar juntos para coletar todas as moedas douradas enquanto evitam os obstáculos. Cada nível trará desafios diferentes, mas apenas a cooperação os levará a vencer este desafio.

O primeiro nível consiste em um caminho sinuoso com obstáculos com plataformas flutuantes, donuts gigantes,gelados enormes,chupa chups e nuvens de algodão doce.Barbie, sendo corajosa e ágil, liderou o caminho, pulando graciosamente sobre as plataformas. Ken, com sua força e habilidade, a protegeu dos perigos durante esta jornada. Trabalhando em perfeita harmonia, eles conseguiram coletar todas as moedas douradas e alcançaram o portal que os levou ao segundo nível.

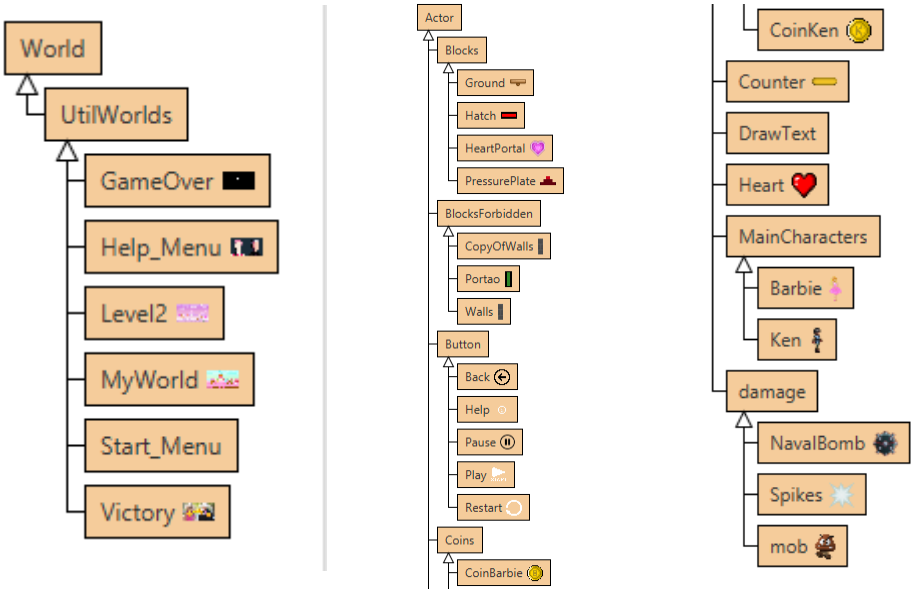
No segundo nível têm de evitar a todo o custo os lacaios dos robôs, os goomba.Têm de evitar todos os obstáculos que apareceram pelo caminho e quem sabe Ken poderá voltar ao estado normal.

Funchal
26 Outubro 2023

Implementação

Primeiramente, ao realizar o projeto optamos por criar um mundo chamado “UtilWorlds”,optamos desta forma pois usamos este mundo apenas como superclasse, desta maneira,usaremos os métodos entre as subclasses do “UtilWorlds” sendo estas: “GameOver”,”Help_Menu”,Level2”,”MyWorld” e “Start_Menu”.

Seguidamente, agrupamos os autores/objetos em “Blocks” sendo as suas subclasses “Ground”,”Hatch”,”HeartPortal” e “PressurePlate”. Em “BlocksForbidden” sendo as suas subclasses “CopyOfWalls”,”Portao”,”Walls”.A classe "Button" contém as subclasses "Back", "Help", "Pause", "Play" e "Restart". "Coins" inclui as subclasses "CoinBarbie" e "CoinKen". "DrawText" engloba as subclasses "BarbieText" e "KenText". Por fim, "MainCharacters" é a superclasse para as personagens principais do jogo, com suas subclasses "Barbie" e "Ken". A classe "damage" contém as subclasses "NavalBomb" e "Spikes".



Funchal
26 Outubro 2023

No “UtilWorlds” decidimos inicializar a maior parte dos objetos,sendo que também foram inicializados alguns objetos nos próprios mundos.Seguidamente usamos o método “addObject(_object_, _x_, _y_)” para adicioná-los nos respectivos mundos.

```
/*Alocar espaço na memoria*/
Restart restart_gameover = new Restart();
Back back_button_helpmenu = new Back(getWidth(),getHeight());

Heart heartken = new Heart();
Heart heartken2 = new Heart();
Heart heartbarbie = new Heart();
Heart heartbarbie2 = new Heart();
Barbie barbie = new Barbie();
Ken ken = new Ken();
private PressurePlate plate = new PressurePlate();
private Portao portao = new Portao();
private Portao portao2 = new Portao();
private Portao portao3 = new Portao();
private Portao portao4 = new Portao();
private Portao portao5 = new Portao();
private PressurePlate plate2 = new PressurePlate();
private PressurePlate plate3 = new PressurePlate();
private PressurePlate plate4 = new PressurePlate();
private PressurePlate plate5 = new PressurePlate();
private Hatch hatch = new Hatch();
Counter actCounter = new Counter("Tempo : ");
Pause pause = new Pause();
CoinBarbie coinBarbie = new CoinBarbie(30);
CoinKen coinKen = new CoinKen(10);
Spikes spikes = new Spikes();
DrawText KenText = new DrawText(Ken,CharactersTextSize,CharactersTextColor,transparent);
DrawText BarbieText = new DrawText(Barbie,CharactersTextSize,CharactersTextColor,transparent);

public static final Color transparent = new Color(0,0,0,0);
public static final String Barbie = "Barbie";
public static final String Ken = "Ken";
public static final int CharactersTextSize=20;
public static final Color CharactersTextColor=Color.MAGENTA;
/*Alocar espaço na memoria*/
```

Para fazer o Score optamos por tratá-lo como uma variável privada dentro do "UtilWorlds" e mostrá-lo no ecrã através de um método chamado "scoreprefix()". Para passar o Score para os outros mundos usamos "getters" e "setters" de maneira a manter o encapsulamento dos dados.

```
public int getScore(){
    return score;
}

public void addScore(int amount){
    score+=amount;
    scoreprefix();
}

public void removeScore(int amount){
    score-=amount;
}

public void scoreprefix()
{
    showText("Score: "+getScore(),50,Height/20);
}
```

Para saber o tempo de jogo fizemos de maneira a encapsular os dados, só que em vez de mostrar o tempo de jogo no ecrã como simplesmente texto, importamos uma classe presente no "Greenfoot" chamada "Counter" cujos autores são "Neil Brown" e "Michael Kölling".Passamos o tempo decorrido para o objeto da classe "Counter".Para saber o tempo decorrido no "Greenfoot" optamos por contar cada frame e sabendo que cada segundo corresponde entre 55 a 60 frames fizemos uma função que realiza esse cálculo e aumenta o tempo decorrido.Como o "timeCounter" a cada frame é incrementado por um eventualmente o resto da divisão vai dar zero.Ou seja no princípio da contagem dos frames quando está no frame zero e nos frames 55 e os seus múltiplos, como por exemplo 110,165 etc

.

```
public void countframes(){
    timeCounter = (timeCounter+1)%55;
        if (timeCounter == 0)
        {
            timeElapsed++;
        }
        actCounter.setValue(timeElapsed + 1);
}
```

Na classe “Coins” usamos mais uma vez o encapsulamento de dados.

```
public class Coins extends Actor
{
    private int Points;
    public void resize_butt(int width,int height){
        GreenfootImage image = getImage();
        image.scale(width, height);
        setImage(image);
    }

    public int getPoints(){
        return Points;
    }
    public void addPoints(int amount){
        Points+=amount;
    }

    public Coins(){
        this.Points=Points;
    }
}
```

O “DrawText” serve unicamente para escrever texto, comportando-se como um objeto.

```
public class DrawText extends Actor
{
    /**
     * Act - do whatever the DrawText wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */

    public DrawText(String s,int size,Color Cletter,Color background){
        setImage(new GreenfootImage(s, size,Cletter,background));
    }
    public void act()
    {
        // Add your action code here.
    }
}
```

Utilizamos uma função que muda o tamanho das imagens,embora com diferentes nomes mas com o mesmo fim na classe “Blocks”,”BlocksForbidden”,”Button” e ”Coins”.

```
    public void resize_image(int x,int y){
        GreenfootImage image = getImage();
        image.scale(x,y);
        setImage(image);
    }
```

Na classe “MainCharacters” inicializamos o seu construtor com os seguintes atributos que serão úteis para ambas as personagens e que determinaram o jogo.São esses os atributos: um inteiro para a velocidade,vida e a altura do salto,um booleano para saber se está virado para a direita, se está a saltar e se está a andar.Realizamos encapsulamento para proteger as variáveis.

```
/*Inicilizacao das personagens*/
public MainCharacters(int Velocity,int Health,boolean FacingRight,boolean Jumping,boolean Walking,int jumpHeight){
    super();
    this.Velocity = Velocity;
    this.Health = Health;
    this.FacingRight = FacingRight;
    this.Jumping = Jumping;
    this.Walking = Walking;
    this.jumpHeight = jumpHeight;
}
```

```
/*Saber se está a andar*/
public boolean isWalking(){
    return Walking;
}

/*Saber a velocidade*/
public int getVelocity(){
    return Velocity;
}

/*Saber para que lado está or
public boolean isFacingRight(
    return FacingRight;
}

/*Saber se está a saltar*/
public boolean isJumping(){
    return Jumping;
}
```

Para realizar o movimento separamos em movimento horizontal,movimento vertical e quando está parado.Se a tecla respetiva a cada personagem para andar for pressionada, a personagem mover-se-á, sendo a sua animação orientada para o lado que move-se,atualizando assim a variável “KeyPressed” tendo assim, a certeza que está a andar.

```
/*ao permir a tecla aciona o movimento,orienta o personagem para o lado em que a tecla está a ser primida.
Atualiza, a variavel KeyPressedFrame para true para depois atualizar a variavel KeyPressed.O que permite que faça
a animacao corretamente*/
public void movement_horinz(String key1, String key2, GreenfootImage[] img_right, GreenfootImage[] img_left) {
boolean KeyPressedFrame = false;
/*variavel auxiliar para mudar o estado de KeyPressed quando nenhuma tecla está a ser primida*/

if (Greenfoot.isKeyDown(key1)) {
    move(+Velocity);
    animation_right(img_right);
    KeyPressedFrame = true;
}
if (Greenfoot.isKeyDown(key2)) {
    move(-Velocity);
    animation_left(img_left);
    KeyPressedFrame = true;
}

// Atualize a variável KeyPressed com o estado atual das teclas
KeyPressed = KeyPressedFrame;
}
```

```
/*Se está a ser pressionada a tecla para saltar é mudado o estado do spaceDown
   para true.Se o objeto estiver no chao(Ground.class) entao pode saltar
   sendo vSpeed uma valor negativo e depois é invocado a funcao para cair*/
public void movement_vertical(String key3) {
if (spaceDown != Greenfoot.isKeyDown(key3)) {//W
    spaceDown = !spaceDown;
    if (spaceDown && onGround(Ground.class))
        vSpeed = jumpHeight;
    fall();
}

}
```

```
/*Combina o movimento horinzontal,vertical e quando a personagem está parada*/
public void movement(String key1,String key2,String key3, GreenfootImage[] img_right, GreenfootImage[] img_left){
    movement_horinz(key1,key2,img_right,img_left);
    movement_vertical(key3);
    movement_stand(img_right,img_left);

}
```

Para a animação de ambos foi necessário fazer dois “array” para cada um, a fim de guardar as várias posições do movimento,um array para quando está voltado para a esquerda e outro para a direita.Para saber se a personagem encontra-se parada apenas é necessário saber se “KeyPressed” é falso, se estiver parado a personagem ainda precisa de saber para que lado está virada. Optamos por deixar a personagem orientada para a última tecla pressionada ao movimentar-se recorrendo à variável “FacingRight”.

```
/*Se nao estiver nenhum dos botoes de movimento a ser usado, entao
   o objeto nao está a andar. Se estiver virado para a direita fica com
   a imagem em que está parado para a direita caso contrario, a outra imagem
   virado para a esquerda*/
public void movement_stand(GreenfootImage[] img_right, GreenfootImage[] img_left){

if (KeyPressed==false) {
Walking = false;
    if (FacingRight) {
        setImage(img_right[0]);
    } else {
        setImage(img_left[0]);
    }
}
}
```

Para fazer aumentar o score as personagens necessitam de apanhar moedas,fizemos com que a função “CoinColide(Class c)”,trate de observar se a personagem apanhou a sua moeda.Atualizando desta forma o score e eliminando a moeda.

```
/*Recebe o coin de cada personagem.Guarda na variavel aMyWorld o mundo atual.
Se a Barbie ou o Ken estiver a tocar no coin faz um cast para coins,atualiza o score e remove o coin do mundo*/
public void CoinColide(Class c){
    UtilWorlds aMyWorld= (UtilWorlds) getWorld();
    if(isTouching(c)){
        GreenfootSound coinmusic = new GreenfootSound("coinsound.mp3");
        coinmusic.play();
        Coins coin_bonus = (Coins) getOneIntersectingObject(c);
        aMyWorld.addScore(coin_bonus.getPoints());
        aMyWorld.removeObject(coin_bonus);
    }
}
```

A função “opendoor” recebe o botão para o abrir e recebe também o portão que o está associado.Se a personagem estiver a tocar no botão o portão/alçapão está aberto e quando a personagem já não se encontra a tocar no botão ele fecha-se.Realizamos “overload” no método “opendoor” sendo que num dos métodos é para abrir o portão e na outra é para abrir o alçapão.

```
//@overload

boolean open = false;
public void opendoor(PressurePlate B, Portao D){
    int X = D.getX();//coordenadas da porta
    int Y = D.getY();
    //A variavel lastopeneddoor vai evitar que outras portas se movimentem no if abaixo
    while(intersects(B) && !open && lastopeneddoor == null){//Se o personagem intersetar com a placa atribuida a porta, e este estiver fechado e o booleano lastopeneddoor for null
        D.setLocation(X, Y-110);//Vai mover a porta para cima
        open= true; //muda a variavel open para true
        lastopeneddoor = D;//e a ultima porta aberta passa para a porta dada como atributo
    }
    if(lastopeneddoor == D && open && !intersects(B)){//Se a ultima porta aberta for a dada, e estiver aberto e, o personagem ja nao esta na placa
        D.setLocation(X,Y+110); //move a porta para baixo
        lastopeneddoor = null; // coloca esta variavel a null
        open = false; // e coloca open como false
    }

}
```

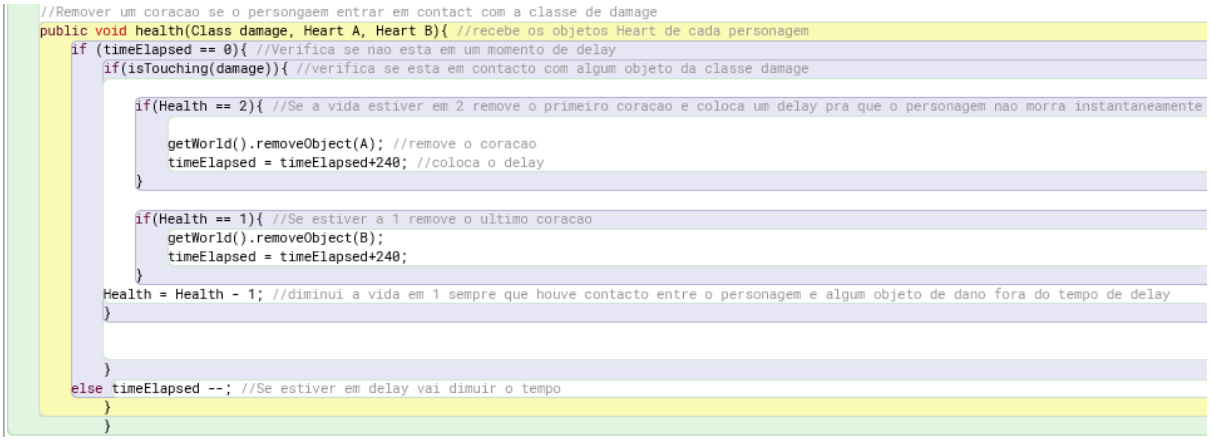
```
public void opendoor(PressurePlate B, Hatch D){
 int X = D.getX(); //coordenadas do Alcapao
 int Y = D.getY();
 //A variavel isbeingopen vai evitar que o alcapao seja aberto duas vezes caso ambos os personagens toquem na placa
 // A variavel lastopenedhatch vai evitar que outros alcapoes se movimentem no if abaixo
 if(D.isbeingopen() == false && intersects(B) && !open && lastopenedhatch == null){ //Se o personagem intersetar com a placa atribuida ao alcapao, e este estiver fecha
     GreenfootSound pressureplatemusicon = new GreenfootSound("pressureplateon.mp3");
     pressureplatemusicon.play();
     D.setLocation(X+110, Y); //Vai mover o alcapao para a direita
     D.changebeingopen();
     open= true; //muda a variavel open para true
     lastopenedhatch = D;    //e o ultimo alcapao aberto passa para o alcapao dado como atributo
 }
 if(lastopenedhatch == D && open && !intersects(B)){ //Se o ultimo alcapao aberto for o dado, e estiver aberto e, o personagem ja nao esta na placa
     GreenfootSound pressureplatemusicout = new GreenfootSound("pressureplateout.mp3");
     pressureplatemusicout.play();
     D.setLocation(X-110,Y); //move o alcapao para a esquerda
     D.changebeingopen();
     lastopenedhatch = null; // coloca esta variavel a null
     open = false; // e coloca open como false
 }

}
```

A função “opendoorslevel1()” faz a associação dos botões às respectivas portas.

```
public void opendoorslevel1(){ //Verifica todas as portas e placas do nivel para ver se as tem de abrir
    MyWorld world = (MyWorld) getWorld();
    opendoor(world.getPlate(1),world.getPortao(5));
    opendoor(world.getPlate(2),world.getPortao(1));
    opendoor(world.getPlate(3), world.getPortao(4));
    opendoor(world.getPlate(4), world.getHatch(1));
}
```

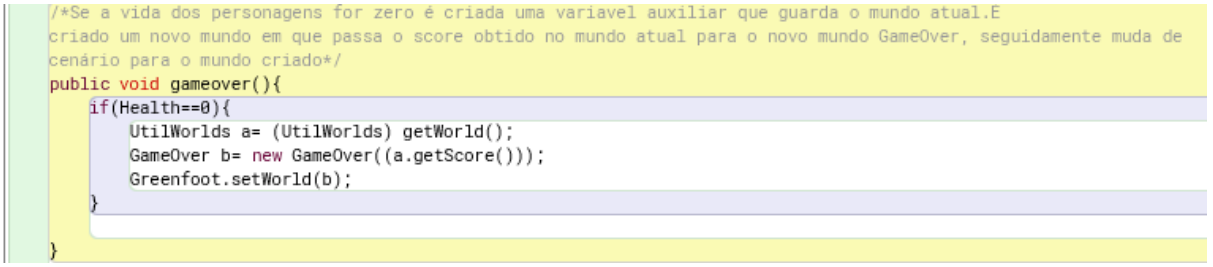
A função “health(Class damage, Heart A, Heart B)” serve para diminuir a vida das personagens se estas tocarem em algum obstáculo, removendo a sua vida e tirando um

coração.Ao perderem um coração a personagem não perderá outro coração no intervalo de quatro segundos.



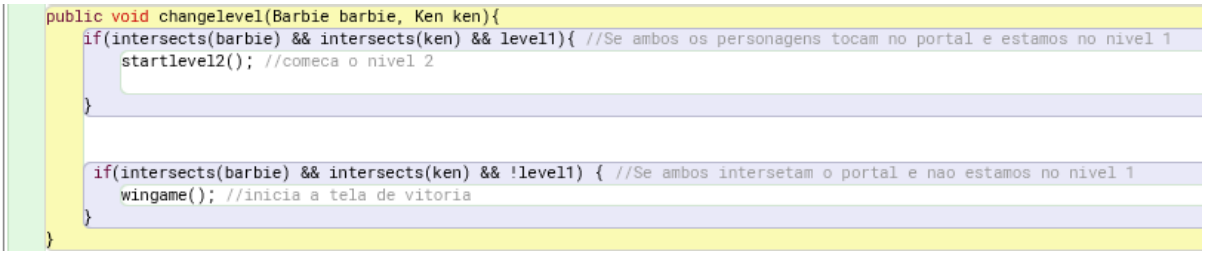
```
//Remover um coracao se o persongaem entrar em contact com a classe de damage
public void health(Class damage, Heart A, Heart B){ //recebe os objetos Heart de cada personagem
    if (timeElapsed == 0){ //Verifica se nao esta em um momento de delay
        if(isTouching(damage)){ //verifica se esta em contacto com algum objeto da classe damage

            if(Health == 2){ //Se a vida estiver em 2 remove o primeiro coracao e coloca um delay pra que o personagem nao morra instantaneamente

                getWorld().removeObject(A); //remove o coracao
                timeElapsed = timeElapsed+240; //coloca o delay
            }

            if(Health == 1){ //Se estiver a 1 remove o ultimo coracao
                getWorld().removeObject(B);
                timeElapsed = timeElapsed+240;
            }
        Health = Health - 1; //diminui a vida em 1 sempre que houve contacto entre o personagem e algum objeto de dano fora do tempo de delay
        }


        }
    else timeElapsed --; //Se estiver em delay vai dimuir o tempo
        }
        }
```

A função “gameover()” serve para quando a vida da personagem encontrar-se a zero mostrar aos jogadores uma mensagem de que perdeu o jogo com o score que obteve e com um botão de restart.



```
/*Se a vida dos personagens for zero é criada uma variavel auxiliar que guarda o mundo atual.E
criado um novo mundo em que passa o score obtido no mundo atual para o novo mundo GameOver, seguidamente muda de
cenário para o mundo criado*/
public void gameover(){
    if(Health==0){
        UtilWorlds a= (UtilWorlds) getWorld();
        GameOver b= new GameOver((a.getScore()));
        Greenfoot.setWorld(b);
    }

}
```

Fizemos um portal para atravessar de um nível para outro ou para ganhar no caso de ser o último nível “changelevel(Barbie barbie, Ken ken)”.As personagens precisam de estar simultaneamente no portal.As funções para mudar de nível ou ganhar são semelhantes apenas diferem no mundo que será mostrado aos jogadores.



```
public void changelevel(Barbie barbie, Ken ken){
    if(intersects(barbie) && intersects(ken) && level1){ //Se ambos os personagens tocam no portal e estamos no nivel 1
        startlevel2(); //comeca o nivel 2

    }


    if(intersects(barbie) && intersects(ken) && !level1) { //Se ambos intersetam o portal e nao estamos no nivel 1
        wingame(); //inicia a tela de vitoria
    }
}
```

Realizamos “overide” na subclasse “GameOver” no método “scoreprefix()” ao receber coordenadas diferentes da classe do pai embora que este “overide” pudesse ser resolvido se o método recebe-se as respectivas coordenadas.

```
//@Overide
public void scoreprefix() //Mostra o Score numa posicao diferente da funcao inicial
{
    showText("Score: "+score,550,300);
}
```

Para fazermos a colisão com as paredes e portões criamos três subclasses de BlocksForbidden que eram CopyOfWalls,Portao e Walls.Fizemos uma função que impedia de a personagem de atravessar qualquer uma subclasse de BlocksForbidden pela esquerda, direita e também que impedia-a de permanecer em cima de uma destas subclasses.Verificamos se o ator estava a colidir com um objeto da classe dos blocos que não podia tocar,depois definiamos “BlocksForbidden c_block = (BlocksForbidden) getOneIntersectingObject(c);” que neste caso era um objeto da classe BlocksForbidden que estava a colidir com o ator e o resultado era armazenado na variável “c_block”.Depois obtivemos a posição destes blocos,a largura do ator e a largura do portão que foi o que nos baseámos para as colisões.Depois usamos um if para verificar se a posição do ator era menor do que o objeto da classe BlocksForbidden e que caso verificasse o ator estava a esquerda deste.Usámos outro if só que se a posição do ator fosse maior do que a do BlocksForbidden dava que ele estava à direita deste.Depois no final definimos a posição da personagem.

```
public void checkCollisionWithBlockForbidden(Class c) {
// Verifica se o ator está a colidir com um objeto da classe dos BlocksForbidden
if (isTouching(c)){//Verificamos se o ator estava a colidir com um objeto da classe dos blocos que não podia tocar
    BlocksForbidden c_block = (BlocksForbidden) getOneIntersectingObject(c);// Um objeto da classe BlocksForbidden que está a colidir com o
    if (c_block != null) {//Se houver colisão
        int newX = getX();// Dá-nos a posição atual da personagem
        int actorWidth = getImage().getWidth(); // Largura do ator
        int c_blockX = c_block.getX(); // Obtemos a posição X do bloco
        int portaoWidth = c_block.getImage().getWidth(); // Obtemos a largura do Portao
        if (getX() < c_blockX) {
            // A personagem está à esquerda do bloco
            newX = c_blockX - (actorWidth / 2) - (portaoWidth / 2);
        } else {
            // A personagem está à direita do bloco
            newX = c_blockX + (portaoWidth / 2) + (actorWidth / 2);
        }

        // Definimos a nova posição da personagem
        setLocation(newX, getY());
    }
}
}
```

Usámos o CheckCollisionWhithGround que era para detectar colisões com a classe ground e também com a classe Hatch. Usamos um if para verificar se a vertical speed que é o nosso vSpeed fosse maior do que 0 , o nosso ator estava descendo e movia-o para cima até que a colisão fosse resolvida e também o vSpeed passava a ser 0.Usámos um if caso o vSpeed fosse menor do que 0 ou seja estava a subir movemos para baixo e definimos o vSpeed =1 para dizer que ele estava a descer.

```
public void checkCollisionWithGround(Class c)
{
 if (isTouching(c)|| isTouching(Hatch.class)){
 // Colisão com a classe Ground e na classe Hatch
// Impede que o personagem "barbie" passe através da plataforma ao descer
 if(vSpeed>0){// Verifica se está descendo
  while (isTouching(c)|| isTouching(Hatch.class)){
      setLocation(getX(), getY() - 1);// Move o personagem para cima até que a colisão seja resolvida
      vSpeed = 0;// Impede o personagem de continuar a cair
      Jumping=false;
        }
    }
 if (vSpeed < 0){//Verifica se está a subir
     while(isTouching(c)){
        setLocation(getX(), getY() + 1);//O ator é movido para baixo
        vSpeed = 1;//vSpeed = 1 logo o nosso ator está a descer
    }
    }
  }
}
```

Fizemos a função checkFalling que chamava outra função que era onGround para verificar se um objeto estava no chão ou não e caso não estivesse chamava uma função que era fall para este objeto cair.

```
public void checkFalling()
{
    //Esta função chama onGround para verificar se o objeto está no chão
    if((onGround(Ground.class)) == false)
    {
        fall();//se não estiver no chão chama a função fall para cair
    }
}
```

O booleano onGround verifica se existe um objeto dessa classe abaixo da personagem.Depois verifica se existe um objeto da classe c que neste caso é Blocks por baixo do actor under Actor under = getOneObjectAtOffset(0,getImage ().getHeight()/2,c); o getImage ().getHeight()/2 é metade da altura do objeto da classe Blocks .No final retorna verdadeiro caso exista um objeto por baixo da personagem.

Usamos a função fall responsável por simular a queda das nossas personagens . Atualizamos a posição vertical , depois incrementamos o nosso vSpeed em 4 unidades e por fim chamamos a função onGround para verificar se estava a colidir com o chão.Se estivesse a colidir ajustaremos a posição vertical para impedir que a nossa personagem ficasse dentro do chão.

```
private boolean onGround(Class c)
{// Esta função verifica se há um objeto da classe c diretamente abaixo do ator.
 // O getImage ().getHeight()/2 é metade da altura do objeto da classe Blocks
Actor under = getOneObjectAtOffset(0,getImage ().getHeight()/2,c);
// Retorna verdadeiro se houver algum objeto em baixo do ator
return under!=null;
}
```

```
rivate void fall()
{
setLocation(getX(),getY() + vSpeed);// A posição vertical é atualizada
vSpeed = vSpeed + 4;//incrementamos 4 ao valor da velocidade vertical

if(onGround(Ground.class)){ // Verifica se o objeto da classe Ground está no chão usando a função 'onGround'.
    setLocation(getX(),getY() - jumpHeight); // Definimos a poscição verical para impedirmos que a nossa personagem fique dentro do chão
}

}
```

Conclusão

No final deste projeto , chegamos à conclusão que foi um trabalho muito árduo e conseguimos impor os conhecimentos obtidos em aula neste projeto . Durante este trabalho ainda deparamo-nos em resolver alguns problemas quanto à implementação do código pois estavam a dar-nos alguns erros na compilação mas tudo foi resolvido . Um dos maiores problemas que tivemos foi os problemas de cast que tivemos algumas dificuldades em resolvê-los .Tivemos que delinear uma ideia quanto ao nosso estilo de jogo que íamos fazer ,debatemos entre o nosso grupo e chegámos a uma conclusão conjunta . Achamos que fizemos um ótimo trabalho para a primeira vez que programamos neste tipo de linguagem , sentimos que o programa, neste caso o greenfoot, ajudou imenso pois enquanto programamos víamos as nossas personagem a implementar o que fazíamos . Por fim, um dos fatores chaves para o desfecho deste trabalho foi a boa distribuição de tarefas e o trabalho de grupo que prevaleceu em todo o projeto.

Funchal
26 Outubro 2023

Gerado pelo “ChatGPT”:
https://chat.openai.com/share/eb65900e-f3f8-44d4-a23f-7b1cbe66b9ae

Pergunta 1: Escreve uma história, detalhada em que o Barbie e a Ken têm de trabalhar em conjunto ao completar um parkour, conseguindo moedas, tentando não acertar em nenhum dos obstáculos.Sendo que em todos os níveis trabalham em cojunto.No primeiro nível ambos passam por um portal e no segundo nível têm de realizar outro parkour para chegar ao portal e completar o jogo.

Resposta:Era uma tarde ensolarada na cidade de Brinquedópolis, onde Barbie e Ken, o casal mais famoso do mundo dos brinquedos, estavam ansiosos para enfrentar um desafio emocionante. Eles receberam um convite para participar de um parkour mágico que os levaria através de diferentes mundos e desafios. Sem hesitar, Barbie e Ken aceitaram a missão e se encontraram diante de um enorme portal cintilante. O portal os transportou para um lugar de paisagens fantásticas, onde árvores gigantes pareciam tocar o céu e riachos de caramelo fluíam sob seus pés. Logo perceberam que estavam no primeiro nível do parkour, e uma voz misteriosa ecoou no ar, explicando as regras do desafio. "Barbie e Ken, bem-vindos ao Parkour dos Sonhos. Vocês devem trabalhar juntos para coletar todas as moedas douradas enquanto evitam os obstáculos. Cada nível trará desafios diferentes, mas apenas a cooperação os levará ao portal do próximo nível." O primeiro nível consistia em um caminho sinuoso com obstáculos como pedras flutuantes, paredes que se moviam e fontes borbulhantes de caramelo. Barbie, sendo corajosa e ágil, liderou o caminho, pulando graciosamente sobre as pedras e se equilibrando nas paredes em movimento. Ken, com sua força e habilidade, a protegeu dos perigos, afastando as fontes de caramelo que jorravam com força. Trabalhando em perfeita harmonia, eles conseguiram coletar todas as moedas douradas e alcançaram o portal que os levou ao segundo nível. No segundo nível, Barbie e Ken se encontraram em um mundo completamente novo. Este era um reino de montanhas flutuantes e nuvens de algodão-doce. O parkour consistia em saltar de montanha em montanha, evitando tempestades de algodão-doce que ameaçavam sugar tudo ao seu redor. A voz misteriosa ecoou novamente: "Aqui, Barbie e Ken, sua cooperação é ainda mais vital. As montanhas flutuantes são instáveis, e a única maneira de alcançar o portal é trabalhando juntos e saltando sincronizados." Barbie e Ken perceberam que deveriam se comunicar e sincronizar seus saltos para superar os desafios. Barbie saltou de

uma montanha para a outra, enquanto Ken a seguiu de perto, lançando seu corpo para impulsioná-la em segurança quando uma tempestade de algodão-doce se aproximou. Era um espetáculo de graça e força, à medida que avançavam de montanha em montanha. Após muitos saltos e desafios superados, Barbie e Ken finalmente chegaram ao portal no topo da última montanha flutuante. Eles olharam um para o outro, com um sorriso de realização e orgulho. "Conseguimos, Ken! Juntos, somos imbatíveis", disse Barbie, abraçando seu parceiro. "Isso mesmo, Barbie. Nada pode nos deter quando trabalhamos juntos", respondeu Ken, retribuindo o abraço. E assim, Barbie e Ken provaram que a cooperação, o trabalho em equipe e a confiança mútua eram as chaves para superar qualquer desafio. Com um brilho nos olhos, eles entraram no portal final, ansiosos para o próximo parkour e as aventuras que os aguardavam no Parkour dos Sonhos.

Anexo

```
import greenfoot.*;

/**
 * Write a description of class UtilWorlds here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class UtilWorlds extends World
{
    /*Alocar espaço na memoria*/
    Restart restart_gameover = new Restart();
    Back back_button_helpmenu = new Back(getWidth(),getHeight());

    Heart heartken = new Heart();
    Heart heartken2 = new Heart();
    Heart heartbarbie = new Heart();
    Heart heartbarbie2 = new Heart();
    Barbie barbie = new Barbie();
    Ken ken = new Ken();
    private PressurePlate plate = new PressurePlate();
    private Portao portao = new Portao();
    private Portao portao2 = new Portao();
    private Portao portao3 = new Portao();
    private Portao portao4 = new Portao();
    private Portao portao5 = new Portao();
    private PressurePlate plate2 = new PressurePlate();
    private PressurePlate plate3 = new PressurePlate();
    private PressurePlate plate4 = new PressurePlate();
    private PressurePlate plate5 = new PressurePlate();
    private Hatch hatch = new Hatch();
    Counter actCounter = new Counter("Tempo : ");
    CoinBarbie coinBarbie = new CoinBarbie(30);
    CoinKen coinKen = new CoinKen(10);
    Spikes spikes = new Spikes();
    DrawText BarbieText = new DrawText(Barbie,CharactersTextSize,CharactersTextColor,transparent);
    DrawText KenText = new DrawText(Ken,CharactersTextSize,CharactersTextColor,transparent);

    public static final Color transparent = new Color(0,0,0,0);
    public static final String Barbie = "Barbie";
    public static final String Ken = "Ken";
```

```
public static final int CharactersTextSize=20;
public static final Color CharactersTextColor=Color.MAGENTA;
/*Alocar espaço na memoria*/
int Height= getHeight();
int Width = getWidth();
private int timeCounter;
private int timeElapsed;
private int score=0;
private String prefix="Score: ";



public void placewalls(){ //coloca as paredes em volta do mundo
    for(int i=0; i<=23; i++){
        addObject(new Walls(), 1096, 579 - (i*25));
    }
    for(int i=0; i<=23; i++){
        addObject(new Walls(), 4, 579 - (i*25));
    }
    for(int i=0; i<=44; i++){
        addObject(new CopyOfWalls(), 20 + (i * 25), 4);
    }

}
public void addPlatform(){ //coloca o chao
    for(int i= 0; i <=36;i++){
        addObject(new Ground(),15 + (i * 30), 598);
    }

}
public void setimelapsed(int value){
    timeElapsed+=value;
}
public int getScore(){
    return score;
}

public void addScore(int amount){
    score+=amount;
    scoreprefix();
}

public void removeScore(int amount){
    score-=amount;
}

public void scoreprefix()
{
    showText("Score: "+getScore(),50,Height/20);
}
public void countframes(){
    timeCounter = (timeCounter+1)%55;
        if (timeCounter == 0)
        {
            timeElapsed++;

        }
        actCounter.setValue(timeElapsed + 1);
}
public PressurePlate getPlate(int n){ //retorna a placa desejada
    switch(n){
        case 1:
            return plate;
        case 2:
            return plate2;
```

```
            case 3:
                return plate3;
            case 4:
                return plate4;
            case 5:
                return plate5;
        }
        return null;

    }
    public Portao getPortao(int n){ //retorna o portao desejado
        switch(n){
            case 1:
                return portao;
            case 2:
                return portao2;
            case 3:
                return portao3;
            case 4:
                return portao4;
            case 5:
                return portao5;
        }
        return null;

    }
    public Hatch getHatch(int n){ //retorna o alcapao desejado
        switch(n){
            case 1:
                return hatch;
        }
        return null;

    }
    public void addMainCharacters(){ //coloca ambos os personagens principais no mundo
        addObject(barbie,1057,548);
        addObject(ken,32,548);
    }
    public UtilWorlds(){
        super(1100, 600, 1);

    }
}
```

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

/**
 * Write a description of class Help_Menu here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Help_Menu extends UtilWorlds
{
    private final int DELTA=60;
    /**
     * Constructor for objects of class Help_Menu.
     *
     */

    private void placeObjects(){
```

```
        addObject(back_button_helpmenu,
        DELTA,
        DELTA
        );
    }
    public Help_Menu()
    {
        super();
        placeObjects();
    }
}

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

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


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

public class GameOver extends UtilWorlds
{
    private int score = 0;
    public GameOver(int score) //recebe o score final como atributo
    {
        super();
        this.score = score;
        prepare();

    }

    //@Overide
    public void scoreprefix() //Mostra o Score numa posicao diferente da funcao inicial
    {
        showText("Score: "+score,550,300);
    }

    private void prepare()
    {
        addObject(restart_gameover,550,450);
        scoreprefix();
    }
}



/**
 * Write a description of class Help_Menu here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Help_Menu extends UtilWorlds
{
    private final int DELTA=60;
    /**
```

```
     * Constructor for objects of class Help_Menu.
     *
     */

    private void placeObjects(){
        addObject(back_button_helpmenu,
        DELTA,
        DELTA
        );
    }
    public Help_Menu()
    {
        super();
        placeObjects();
    }
}



/**
 * Write a description of class HistoryScreen here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class HistoryScreen extends UtilWorlds
{
    /**
     * Constructor for objects of class HistoryScreen.
     *
     */
    Play playgame = new Play(getWidth(),getHeight());
    GreenfootSound barbieminaj = new GreenfootSound("plsmusic.mp3");
    public HistoryScreen()
    {
        addObject(playgame, 550, 500);
        barbieminaj.play();


    }
    public void act(){
        playgame.StartGame();
        if(playgame.StartGame()){
            barbieminaj.stop();
        }
    }
}

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

/**
 * Write a description of class Level2 here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Level2 extends UtilWorlds
{
    private int score;
    private int StartingVelocity=5;
    private int timeElapsed;
    /**
     * Constructor for objects of class Level2.
     *
     */
```

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


public Heart getHeart(int n){ //retorna o coracao desejado
    switch(n){
        case 0:
            return heartken;
        case 1:
            return heartken2;
        case 2:
            return heartbarbie;
        case 3:
            return heartbarbie2;
    }
    return null;

}

private void placeObjects(){ //coloca todos os objetos individuais deste mundo no mundo
    int worldWidth = getWidth();
    int worldHeight = getHeight();
    addObject(new Walls(),175, 580);
    addObject(new Walls(), 175, 555);
    addObject(new Ground(), 195, 550);
    addObject(new Ground(), 225, 550);
    addObject(new Ground(), 255, 550);
    addObject(new Ground(), 285, 550);
    addObject(new Ground(), 315, 550);
    addObject(new Walls(), 330, 580);
    addObject(new Walls(), 330, 555);
    addObject(new Walls(), 330, 530);
    addObject(new Walls(), 330, 510);
    addObject(new mob(), 930, 480);
    addObject(new Ground(), 350, 500);
    addObject(new Ground(), 380,500);
    addObject(new Ground(), 410,500);
    addObject(new Ground(), 440,500);
    addObject(new Ground(), 470,500);
    addObject(new Ground(), 500,500);
    addObject(new Ground(), 530,500);
    addObject(new Walls(), 550, 580);
    addObject(new Walls(), 550, 555);
    addObject(new Walls(), 550, 530);
    addObject(new Walls(), 550, 510);
    addObject(new Ground(), 585, 470);
    addObject(new Ground(), 630, 425);
    addObject(new Ground(), 760, 450);
    addObject(new Ground(), 655, 390);
    addObject(new Ground(), 685, 390);
    addObject(new Ground(), 715, 390);
    addObject(new Ground(), 535, 390);
    addObject(new Ground(), 505, 390);
    addObject(new Ground(), 475, 390);
    addObject(new Ground(), 445, 390);
    addObject(new Ground(), 415, 390);
    addObject(new Walls(), 815, 580);
    addObject(new Walls(), 815, 555);
    addObject(new Walls(), 815, 530);
    addObject(new Walls(), 815, 510);
    addObject(new Ground(), 835, 500);
    addObject(new Ground(), 865, 500);
    addObject(new Ground(), 895, 500);
    addObject(new Ground(), 925, 500);
```

```
    addObject(new Ground(), 955, 500);
    addObject(new Walls(), 970, 530);
    addObject(new Walls(), 970, 510);
    addObject(new Ground(),980, 550);
    addObject(new Ground(),1010, 550);
    addObject(new Walls(), 1025, 580);
    addObject(new Walls(), 1025, 555);
    addObject(new Ground(), 780, 300);
    addObject(new Ground(), 810, 300);
    addObject(new Ground(), 840, 300);
    addObject(new Ground(), 870, 300);
    addObject(new Ground(), 900, 300);
    addObject(new Ground(), 930, 300);
    addObject(new Ground(), 960, 300);
    addObject(new Ground(), 610, 310);
    addObject(new Ground(), 580, 310);
    for(int i=0; i<7; i++){
    addObject(new Ground(), 530 - (30*i), 300);
    }
    addObject(new Walls(), 335, 285);
    addObject(new Walls(), 335, 265);
    for(int i=0; i <5; i++){
       addObject(new Ground(), 315 - (30 * i), 255);
    }
    addObject(new Walls(), 180, 215);
    addObject(new Walls(), 180, 240);
    addObject(new Ground(), 170, 205);
    addObject(new Ground(), 140, 205);
    addObject(new Walls(), 120, 190);
    addObject(new Walls(), 120, 165);
    addObject(new Walls(), 120, 140);
    addObject(new Walls(), 120, 115);
    for(int i = 0; i < 15; i++){
       addObject(new Ground(), 260 + (i*30),145);
    }
    addObject(new mob(), 515, 367);
    addObject(new mob(), 790, 575);
    addObject(new mob(), 690, 575);
    addObject(new mob(), 300, 530);
    addObject(new Spikes(), 640, 500);
    addObject(new NavalBomb(), 450, 205);
    addObject(new CoinBarbie(30),470,115);
    addObject(new CoinKen(30),525,115);
    addObject(new CoinBarbie(30),550,260);
    addObject(new CoinKen(30),880,260);
    addObject(new HeartPortal(false),310,95);

    addObject(heartken,94,80);
    addObject(heartken2,50,80);
    addObject(heartbarbie,getWidth()-94,80);
    addObject(heartbarbie2,getWidth()-50,80);
    addObject(BarbieText,1030,56);
    addObject(KenText,70,56);


    actCounter = new Counter("Tempo: ");
    addObject(actCounter, 550, worldHeight/20);
}

public void prepare(){
    addPlatform();
    placewalls();
    placeObjects();
    addMainCharacters();
} //coloca todos os objetos necessarios para o bom funcionamento do jogo
```

```
    public Level2(int score,int value) //recebe o score do mundo anterior e o tempo para continuar de onde parou
    {
        super();
        addScore(score);
        setimelapsed(value);
        prepare();
    }
}


import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)
import java.util.List;
/**
 * Write a description of class MyWorld here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class MyWorld extends UtilWorlds
{

    public MyWorld()
    {
        // Create a new world with 1100x600 cells with a cell size of 1x1 pixels.
        super();
        prepare();
        scoreprefix();
    }

    private void placeObjects() { //coloca todos os objetos individuais deste mundo no mundo
        int worldWidth = getWidth();
        int worldHeight = getHeight();
        addObject(new Ground(),160,300);
        addObject(new Ground(),190,300);
        addObject(new Ground(),130,300);
        addObject(new Ground(),220,300);
        addObject(new Ground(),310,260);
        addObject(new Ground(),430,265);
        addObject(new Ground(),540,240);
        addObject(new Ground(),350,400);
        addObject(new Ground(),615,300);
        addObject(new Ground(),705,240);
        addObject(new Ground(),780,275);
        addObject(new Ground(),280,355);
        addObject(new Ground(),840,325);
        addObject(new Ground(),980,400);
        addObject(new Ground(),925,360);
        addObject(new NavalBomb(),750,300);
        addObject(new NavalBomb(),160,250);
        addObject(new Ground(),340,260);
        addObject(new Ground(),645,300);
        addObject(new Ground(),505,260);
        addObject(getPortao(1),185,535);
        addObject(getPortao(2),490,535);
        addObject(getPortao(3),490, 415);
        addObject(getPortao(4), 695, 415);
        addObject(getPortao(5), 840, 535);
        addObject(getPlate(1),110, 585);
        addObject(getPlate(2),770,585);
        addObject(getPlate(3), 740, 465);
        addObject(getPlate(4),1060,465);
        addObject(getHatch(1), 620, 235);
        addObject(new HeartPortal(true),90,430);
        addObject(new Spikes(), 435, 375);
```

```
        for(int i= 0; i <=13;i++){
            addObject(new Ground(),15 + (i * 30), 480);
        }
        for(int i= 23; i <=36;i++){
            addObject(new Ground(),15 + (i * 30), 480);
         }
        addObject(new CoinBarbie(30),630,270);
        addObject(new CoinKen(30),800,410);
        addObject(heartken,94,80);
        addObject(heartken2,50,80);
        addObject(heartbarbie,getWidth()-94,80);
        addObject(heartbarbie2,getWidth()-50,80);
        addObject(BarbieText,1030,56);
        addObject(KenText,70,56);


        actCounter = new Counter("Tempo: ");
        addObject(actCounter, 550, worldHeight/20);

    }



    public void prepare() //coloca todos os objetos necessarios para o bom funcionamento do jogo
    {
        addPlatform();
        placewalls();
        placeObjects();
        addMainCharacters();

    }

    public Heart getHeart(int n){ //retorna o coracao desejado
        switch(n){
            case 0:
                return heartken;
            case 1:
                return heartken2;
            case 2:
                return heartbarbie;
            case 3:
                return heartbarbie2;
        }
        return null;

    }

    public void act()
    {
        countframes();
    }

}



/**
 * Write a description of class Start_Menu here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Start_Menu extends UtilWorlds
{
private final int DELTA=110;//valor que nao muda em vez de mudar em todo o lado
```

```
    /**
     * Constructor for objects of class Start_Menu.
     *
     */
     private void BackgroundColor(){
        GreenfootImage background = getBackground();//Create Image
        background.setColor(Color.BLACK);//Add Background color
        background.fillRect(0,0,getWidth(),getHeight());//Fill image with color
        }
    Play play_button = new Play(getWidth(),getHeight());
     private void placeObjects(){
        int worldWidth = getWidth();//vai buscar o comprimento do mundo(X)
        int worldHeight = getHeight();//vai buscar a altura do mundo(Y)

        //constroi os botoes

        Help help_button = new Help(worldWidth,worldHeight);

        //adiciona os botoes sabendo o nome do objeto e a localizacao em x e y
        addObject(play_button,
        worldWidth/2-DELTA,
        worldHeight/2
        );

        addObject(help_button,
        worldWidth/2+DELTA,
        worldHeight/2
        );


    }
    public Start_Menu()
    {
        // Create a new world with 600x400 cells with a cell size of 1x1 pixels.
        super();
        BackgroundColor();
        placeObjects();

    }
    public void act(){
        play_button.HistoryMenu();
    }
}


   /**
 * Write a description of class Victory here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Victory extends UtilWorlds
{

    /**
     * Constructor for objects of class Victory.
     *
     */
    public Victory(int score,int value) //Mostra as informacoes do jogo
    {
        addObject(restart_gameover,550,450);
        addObject(actCounter,550,100);
        actCounter.setValue(value);
        actCounter.setPrefix("Demorou: ");
        actCounter.setSuffix(" segundos");
```

```
        showText("Score: "+score,50,Height/20);
        GreenfootSound victorymusic = new GreenfootSound("victorymusic.mp3");
        victorymusic.play();
        // if(restart_gameover.restartgame()){
            // victorymusic.stop();
        // }
    }
}
```

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

/**
 * Write a description of class Blocks here.
 *
 * @author (your name)
 * @version (a version number or a date)
/**
 * Write a description of class Blocks here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Blocks extends Actor
{
    /**
     * Act - do whatever the Blocks wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void resize_image(int x,int y){
        GreenfootImage image = getImage();
        image.scale(x,y);
        setImage(image);
    }
    public void act()
    {
        // Add your action code here.
    }
}
```

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

/**
 * Write a description of class Ground here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Ground extends Blocks
{
    /**
     * Act - do whatever the Ground wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        // Add your action code here.
    }
}
```

```
/**
 * Write a description of class Hatch here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Hatch extends Blocks
{
    private boolean isbeingopen = false; //verifica se o alcapao esta aberto no momento
    /**
     * Act - do whatever the Hatch wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public boolean isbeingopen(){ //retorn o valor do booleano
        return isbeingopen;
    }
    public void changebeingopen(){ //altera o valor do booleano para o seu oposto
        isbeingopen = !isbeingopen;
    }
    public Hatch(){
        GreenfootImage Hatch = getImage();
        Hatch.scale(120,15);
    }
    public void act()
    {
        // Add your action code here.
    }
}



import greenfoot.*;  // (World, Actor, GreenfootImage, Greenfoot and MouseInfo)
import java.util.List;

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

public class HeartPortal extends Blocks
{
    /**
     * Act - do whatever the HeartPortal wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    private boolean level1= true; //variavel para saber se ainda estamos no nivel 1
    public HeartPortal(boolean level1){ //quando o portal e criado ele recebe o atributo se e do nivel 1 ou nao
        this.level1 = level1;
        resize_image(80,80);
    }

    public void changelevel(Barbie barbie, Ken ken){
          if(intersects(barbie) && intersects(ken) && level1){ //Se ambos os personagens tocam no portal e estamos
no nivel 1
            startlevel2(); //comeca o nivel 2
            GreenfootSound portalsound = new GreenfootSound("portalsound.mp3");
            portalsound.play();
        }


          if(intersects(barbie) && intersects(ken) && !level1) { //Se ambos intersetam o portal e nao estamos no nivel
1
            wingame(); //inicia a tela de vitoria
```

```
        }
    }

    public void startlevel2(){ //inicia o nivel2
        UtilWorlds a = (UtilWorlds) getWorld();
        MyWorld b = (MyWorld) getWorld();
         UtilWorlds c = new Level2(a.getScore(),b.actCounter.getValue()); //Inicia o nivel 2 com o score do utilworlds,
e o contador de tempo do primeiro mundo
        Greenfoot.setWorld(c); //munda para esse nivel criado
    }
    public void wingame(){
        UtilWorlds a = (UtilWorlds) getWorld();
        Level2 c = (Level2) getWorld();
          UtilWorlds V = new Victory(a.getScore(),c.actCounter.getValue()); //Inicia a tela de vitoria com o score do
utilworlds, e o contador de tempo do jogo
        Greenfoot.setWorld(V); //muda para esta tela
    }
    public void act()
    {
    if(level1){ //Se estiver no nivel 1, usa os personagens do nivel 1 como atributos
        MyWorld a= (MyWorld) getWorld();
        changelevel(a.barbie, a.ken);
    }
    if(!level1){ //Senao usa os do nivel 2
        Level2 b = (Level2) getWorld();
        changelevel(b.barbie,b.ken);
    }
    }
    }


/**
 * Write a description of class PressurePlate here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class PressurePlate extends Blocks
{
    /**
     * Act - do whatever the PressurePlate wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
     /*PROVALVEMENTE É NECESSARIO FAZER DOIS BOTOES, 1 PARA SER A PARTE DE CIMA DO BOTAO
E OUTRO PARA SER AS LATERAIS*/
    public void act()
    {
    }
}


/**
 * Write a description of class BlocksForbidden here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class BlocksForbidden extends Actor
{
    /**
     * Act - do whatever the BlocksForbidden wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void resize_image(int x,int y){
        GreenfootImage image = getImage();
```

```
        image.scale(x,y);
        setImage(image);
    }
    public void act()
    {
        // Add your action code here.
    }
}



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

/**
 * Write a description of class CopyOfWalls here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class CopyOfWalls extends BlocksForbidden
{
    /**
     * Act - do whatever the CopyOfWalls wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
    }
}



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

/**
 * Write a description of class Portao here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Portao extends BlocksForbidden
{
    private boolean isbeingopen = false; //Verifica se a porta esta aberta no momento
    /**
     * Act - do whatever the Portao wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public boolean isbeingopen(){ //retorna o booleano
        return isbeingopen;
    }
    public void changebeingopen(){ //altera o booleano para o seu oposto
        isbeingopen = !isbeingopen;
    }
    public Portao() {
        resize_image(15,120);
    }
    public void act()
    {

    }
}

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

/**
```

```
 * Write a description of class Walls here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Walls extends BlocksForbidden
{
    /**
     * Act - do whatever the Walls wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        // Add your action code here.
    }
}


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

/**
 * Write a description of class Button here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Button extends Actor
{
    /**
     * Act - do whatever the Button wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */

    public void resize_butt(int width,int height){
        GreenfootImage image = getImage();
        image.scale(width, height);
        setImage(image);
    }
    public void act()
    {
        // Add your action code here.
    }
}


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

/**
 * Write a description of class Back here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Back extends Button
{
    /**
     * Act - do whatever the Back wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public Back(int Width,int Height){
        super();
        resize_butt(100,100);
    }
    public void act()
    {
```

```
        if(Greenfoot.mouseClicked(this)){
            Greenfoot.setWorld(new Start_Menu());
        }
    }
}


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

/**
 * Write a description of class Help here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Help extends Button
{
    /**
     * Act - do whatever the Help wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public Help(int Width,int Height){
        super();
        resize_butt(300,300);
    }
    public void act()
    {
        if(Greenfoot.mouseClicked(this)){
            Greenfoot.setWorld(new Help_Menu());
        }
    }
}



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

/**
 * Write a description of class Play here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Play extends Button
{
    /**
     * Act - do whatever the Play wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public Play(int Width,int Height){
        super();
        resize_butt(150,150);
    }
    public void HistoryMenu(){
        if(Greenfoot.mouseClicked(this)){
            Greenfoot.setWorld(new HistoryScreen());
        }
    }
    public boolean StartGame(){
        if(Greenfoot.mouseClicked(this)){
            Greenfoot.setWorld(new MyWorld());
            return true;
        }
        return false;
    }
```

```
    public void act()
    {
    }
}


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 Button
{
    /**
     * Act - do whatever the Restart wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public boolean restartgame(){
        if(Greenfoot.mouseClicked(this)){
            Greenfoot.setWorld(new Start_Menu());
            return true;
        }
        else{
            return false;
        }
    }
    public void act()
    {
        restartgame();
    }
}



import greenfoot.*;

/**
 * Write a description of class Coins here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Coins extends Actor
{
    private int Points;
    public void resize_butt(int width,int height){
        GreenfootImage image = getImage();
        image.scale(width, height);
        setImage(image);
    }

    public int getPoints(){
        return Points;
    }
    public void addPoints(int amount){
        Points+=amount;
    }

    public Coins(){
        this.Points=Points;
    }
}
```

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

/**
 * Write a description of class CoinBarbie here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class CoinBarbie extends Coins
{
    private int Points;

    /**
     * Act - do whatever the CoinBarbie wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public CoinBarbie(int Points){
        super();
        addPoints(Points);
        resize_butt(35,35);
    }
    public void act()
    {
    }
}
```

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

/**
 * Write a description of class CoinKen here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class CoinKen extends Coins
{
    private int Points;
    /**
     * Act - do whatever the CoinKen wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public CoinKen(int Points){
        super();
        addPoints(Points);
        resize_butt(35,35);
    }
    public void act()
    {

    }
}
```

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

/**
 * A Counter class that allows you to display a numerical value on screen.
 *
 * The Counter is an actor, so you will need to create it, and then add it to
 * the world in Greenfoot.  If you keep a reference to the Counter then you
 * can adjust its value.  Here's an example of a world class that
 * displays a counter with the number of act cycles that have occurred:
```

```
 *
 * <pre>
 * class CountingWorld
 * {
 *     private Counter actCounter;
 *
 *     public CountingWorld()
 *     {
 *         super(600, 400, 1);
 *         actCounter = new Counter("Act Cycles: ");
 *         addObject(actCounter, 100, 100);
 *     }
 *
 *     public void act()
 *     {
 *         actCounter.setValue(actCounter.getValue() + 1);
 *     }
 * }
 * </pre>
 *
 * @author Neil Brown and Michael Kölling
 * @version 1.0
 */

/**Classe importada**/

public class Counter extends Actor
{
    private static final Color transparent = new Color(0,0,0,0);
    private GreenfootImage background;
    private int value;
    private int target;
    private String prefix;
    private String suffix;

    public Counter()
    {
        this(new String());
    }

    /**
     * Create a new counter, initialised to 0.
     */
    public Counter(String prefix)
    {
        background = getImage();  // get image from class
        value = 0;
        target = 0;
        this.prefix = prefix;
        this.suffix = suffix;
        updateImage();
    }

    /**
     * Animate the display to count up (or down) to the current target value.
     */
    public void act()
    {
        if (value < target) {
            value++;
            updateImage();
        }
        else if (value > target) {
            value--;
            updateImage();
```

```
    }
}

/**
 * Add a new score to the current counter value.  This will animate
 * the counter over consecutive frames until it reaches the new value.
 */
public void add(int score)
{
    target += score;
}

/**
 * Return the current counter value.
 */
public int getValue()
{
    return target;
}

/**
 * Set a new counter value.  This will not animate the counter.
 */
public void setValue(int newValue)
{
    target = newValue;
    value = newValue;
    updateImage();
}

/**
 * Sets a text prefix that should be displayed before
 * the counter value (e.g. "Score: ").
 */
public void setPrefix(String prefix)
{
    this.prefix = prefix;
    updateImage();
}
public void setSuffix(String suffix)
{
    this.suffix = suffix;
    updateImage();
}

/**
 * Update the image on screen to show the current value.
 */
private void updateImage()
{
    GreenfootImage image = new GreenfootImage(background);
    GreenfootImage text = new GreenfootImage(prefix + value, 22, Color.BLACK, transparent);
    if(suffix!=null){
        text = new GreenfootImage(prefix + value + suffix, 22, Color.BLACK, transparent);
    }
    if (text.getWidth() > image.getWidth() - 20)
    {
        image.scale(text.getWidth() + 20, image.getHeight());
    }

    image.drawImage(text, (image.getWidth()-text.getWidth())/2,
                    (image.getHeight()-text.getHeight())/2);
    setImage(image);
}
```

```
}
```

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

/**
 * Write a description of class DrawText here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class DrawText extends Actor
{
    /**
     * Act - do whatever the DrawText wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */

    public DrawText(String s,int size,Color Cletter,Color background){
        setImage(new GreenfootImage(s, size,Cletter,background));
    }
    public void act()
    {
        // Add your action code here.
    }
}
```

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

/**
 * Write a description of class Heart here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Heart extends Actor
{
    /**
     * Act - do whatever the Heart wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        // Add your action code here.
    }
}
```

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

/**
 * Write a description of class MainCharacters here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class MainCharacters extends Actor
{
    private int Velocity;
    private boolean Walking;
    private boolean FacingRight;
    private boolean Jumping;
    private int vSpeed = 0;
```

```
    private int animationCounter = 0;
    private int frame;
    private boolean spaceDown=false;
    private  int jumpHeight;
    private boolean KeyPressed;
    private MyWorld aMyWorld;


    int pause = 0;
    private int Health=2;
    private int timeCounter;
    public int timeElapsed=0;
    Portao lastopeneddoor = null;
    Hatch lastopenedhatch = null;

    /*Recebe o coin de cada personagem.Guarda na variavel aMyWorld o mundo atual.
      Se a Barbie ou o Ken estiver a tocar no coin faz um cast para coins,atualiza o score e remove o coin do
mundo*/
    public void CoinColide(Class c){
        UtilWorlds aMyWorld= (UtilWorlds) getWorld();
        if(isTouching(c)){
            GreenfootSound coinmusic = new GreenfootSound("coinsound.mp3");
            coinmusic.play();
            Coins coin_bonus = (Coins) getOneIntersectingObject(c);
            aMyWorld.addScore(coin_bonus.getPoints());
            aMyWorld.removeObject(coin_bonus);
        }
    }
    /*Saber se está a andar*/
    public boolean isWalking(){
        return Walking;
    }

    /*Saber a velocidade*/
    public int getVelocity(){
        return Velocity;
    }

    /*Saber para que lado está orientado*/
    public boolean isFacingRight(){
        return FacingRight;
    }

    /*Saber se está a saltar*/
    public boolean isJumping(){
        return Jumping;
    }

    public void act()
    {

    }

    /*Como a animacao é frames por segundo foi necessario fazer um delay para o
          movimento nao ser muito rapido, como sao cerca de 55/60 frames por segundo
          significa que em 11 frames ja está na segunda posicao img[1] e assim
          sucessivamente para as outras posicoes*/
    public void animation_left(GreenfootImage[] img){
        int delay = 10;
        /*está a andar para a esquerda logo está virado para a esquerda*/
        FacingRight = false;
        Walking=true;
        frame ++;

        if(frame < 1 * delay)
```

```
        {
            setImage(img[0]);
        }
        else if(frame < 2 * delay)
        {
            setImage(img[1]);
        }
        else if(frame < 3 * delay)
        {
            setImage(img[2]);
        }
        else if (frame < 4 * delay)
        {
            setImage(img[3]);
            frame = 1;
            return;
        }
    }

    /*Mesma mecanica da funcao anterior so que em vez de ir para a esquerda
          vai para a direita*/
    public void animation_right(GreenfootImage[] img){

        int delay = 10;

        FacingRight = true;
        Walking=true;
        frame ++;
        if(frame < 1 * delay)
        {
            setImage(img[0]);
        }
        else if(frame < 2 * delay)
        {
            setImage(img[1]);
        }
        else if(frame < 3 * delay)
        {
            setImage(img[2]);
        }
        else if (frame < 4 * delay)
        {
            setImage(img[3]);
            frame = 1;
            return;
        }
    }

    /*ao permir a tecla aciona o movimento,orienta o personagem para o lado em que a tecla está a ser primida.
     Atualiza, a variavel KeyPressedFrame para true para depois atualizar a variavel KeyPressed.O que permite
que faça
    a animacao corretamente*/
    public void movement_horinz(String key1, String key2, GreenfootImage[] img_right, GreenfootImage[] img_left)
{
    boolean KeyPressedFrame = false;
    /*variavel auxiliar para mudar o estado de KeyPressed quando nenhuma tecla está a ser primida*/

    if (Greenfoot.isKeyDown(key1)) {
        move(Velocity);
        animation_right(img_right);
        KeyPressedFrame = true;
    }
    if (Greenfoot.isKeyDown(key2)) {
        move(-Velocity);
        animation_left(img_left);
```

```
        KeyPressedFrame = true;
    }

    // Atualize a variável KeyPressed com o estado atual das teclas
    KeyPressed = KeyPressedFrame;
    }

    /*Se nao estiver nenhum dos botoes de movimento a ser usado, entao
      o objeto nao está a andar. Se estiver virado para a direita fica com
      a imagem em que está parado para a direita caso contrario, a outra imagem
      virado para a esquerda*/
    public void movement_stand(GreenfootImage[] img_right, GreenfootImage[] img_left){

    if (KeyPressed==false) {
    Walking = false;
        if (FacingRight) {
            setImage(img_right[0]);
        } else {
            setImage(img_left[0]);
        }
    }
    }

    /*Combina o movimento horinzontal,vertical e quando a personagem está parada*/
        public void movement(String key1,String key2,String key3, GreenfootImage[] img_right, GreenfootImage[]
img_left){
        movement_horinz(key1,key2,img_right,img_left);
        movement_vertical(key3);
        movement_stand(img_right,img_left);

    }

    /*Se está a ser pressionada a tecla para saltar é mudado o estado do spaceDown
      para true.Se o objeto estiver no chao(Ground.class) entao pode saltar
      sendo vSpeed uma valor negativo e depois é invocado a funcao para cair*/
    public void movement_vertical(String key3) {
    if (spaceDown != Greenfoot.isKeyDown(key3)) {//W
        spaceDown = !spaceDown;
        if (spaceDown && onGround(Ground.class))
            vSpeed = jumpHeight;
        fall();
    }

    }

     private void fall()
    {
    setLocation(getX(),getY() + vSpeed);// A posição vertical é atualizada
    vSpeed = vSpeed + 4;//incrementamos 4 ao valor da velocidade vertical

        if(onGround(Ground.class)){  // Verifica se o objeto da classe Ground está no chão usando a função
'onGround'.
            setLocation(getX(),getY() - jumpHeight); // Definimos a poscição verical para impedirmos que a nossa
personagem fique dentro do chão
    }

    }


    private boolean onGround(Class c)
    {// Esta função verifica se há um objeto da classe c diretamente abaixo do ator.
     // O getImage ().getHeight()/2 é metade da altura do objeto da classe Blocks
    Actor under = getOneObjectAtOffset(0,getImage ().getHeight()/2,c);
    // Retorna verdadeiro se houver algum objeto em baixo do ator
    return under!=null;
```

```
    }

    public void checkFalling()
    {
      //Esta função chama onGround para verificar se o objeto está no chão
      if((onGround(Ground.class)) == false)
      {
          fall();//se não estiver no chão chama a função fall para cair
      }
    }

    public void checkCollisionWithGround(Class c)
    {
     if (isTouching(c)|| isTouching(Hatch.class)){
     // Colisão com a classe Ground e na classe Hatch
    // Impede que o personagem "barbie" passe através da plataforma ao descer
     if(vSpeed>0){// Verifica se está descendo
      while (isTouching(c)|| isTouching(Hatch.class)){
          setLocation(getX(), getY() - 1);// Move a barbie ou o ken para cima até que a colisão seja resolvida
          vSpeed = 0;// Impede o ator  de continuar a cair
          Jumping=false;
            }
         }
     if (vSpeed < 0){//Verifica se está a subir
         while(isTouching(c)){
            setLocation(getX(), getY() + 1);//O ator é movido para baixo
            vSpeed = 1;//vSpeed = 1 logo o nosso ator está a descer
       }
       }
      }
    }

    public void checkCollisionWithBlockForbidden(Class c) {
    // Verifica se o ator está a colidir com um objeto da classe dos BlocksForbidden
     if (isTouching(c)){//Verificamos se o ator estava a colidir com um objeto da classe dos blocos que não podia
tocar
              BlocksForbidden c_block = (BlocksForbidden) getOneIntersectingObject(c);// Um objeto da classe
BlocksForbidden que está a colidir com o ator e o resultado é armazenado na variável c_block
        if (c_block != null) {//Se houver colisão
            int newX = getX();// Dá-nos a posição atual da personagem
            int actorWidth = getImage().getWidth(); // Largura do ator
            int c_blockX = c_block.getX(); // Obtemos a posição X do bloco
            int portaoWidth = c_block.getImage().getWidth(); // Obtemos a largura do Portao
            if (getX() < c_blockX) {
                // A personagem está à esquerda do bloco
                newX = c_blockX - (actorWidth / 2) - (portaoWidth / 2);
            } else {
                // A personagem está à direita do bloco
                newX = c_blockX + (portaoWidth / 2) + (actorWidth / 2);
            }

            // Definimos a nova posição da personagem
            setLocation(newX, getY());
        }
    }
    }

    /*Inicilizacao das personagens*/
        public MainCharacters(int Velocity,int Health,boolean FacingRight,boolean Jumping,boolean Walking,int
jumpHeight){
        super();
        this.Velocity = Velocity;
        this.Health = Health;
        this.FacingRight = FacingRight;
        this.Jumping = Jumping;
```

```
        this.Walking = Walking;
        this.jumpHeight = jumpHeight;
    }

    public void opendoorslevel1(){ //Verifica todas as portas e placas do nivel para ver se as tem de abrir
        MyWorld world = (MyWorld) getWorld();
        opendoor(world.getPlate(1),world.getPortao(5));
        opendoor(world.getPlate(2),world.getPortao(1));
        opendoor(world.getPlate(3), world.getPortao(4));
        opendoor(world.getPlate(4), world.getHatch(1));
    }
    //@overload

    boolean open = false;
    public void opendoor(PressurePlate B, Portao D){
        int X = D.getX();//coordenadas da porta
        int Y = D.getY();
         //A variavel isbeingopen vai evitar que a porta seja aberta duas vezes caso ambos os personagens toquem na placa
        //A variavel lastopeneddoor vai evitar que outras portas se movimentem no if abaixo
            if(D.isbeingopen() == false && intersects(B) && !open && lastopeneddoor == null){//Se o personagem intersetar com a placa atribuida a porta, e este estiver fechado e o booleano lastopeneddoor for null
            GreenfootSound pressureplatemusicon = new GreenfootSound("pressureplateon.mp3");
            pressureplatemusicon.play();
            D.setLocation(X, Y-110);//Vai mover a porta para cima
            D.changebeingopen();
            open= true; //muda a variavel open para true
            lastopeneddoor = D;//e a ultima porta aberta passa para a porta dada como atributo
        }
         if(lastopeneddoor == D && open && !intersects(B)){//Se a ultima porta aberta for a dada, e estiver aberto e, o personagem ja nao esta na placa
            GreenfootSound pressureplatemusicout = new GreenfootSound("pressureplateout.mp3");
            pressureplatemusicout.play();
            D.setLocation(X,Y+110); //move a porta para baixo
            D.changebeingopen();
            lastopeneddoor = null; // coloca esta variavel a null
            open = false; // e coloca open como false
        }


    }

       public void opendoor(PressurePlate B, Hatch D){
        int X = D.getX(); //coordenadas do Alcapao
        int Y = D.getY();
            //A variavel isbeingopen vai evitar que o alcapao seja aberto duas vezes caso ambos os personagens toquem na placa
        // A variavel lastopenedhatch vai evitar que outros alcapoes se movimentem no if abaixo
            if(D.isbeingopen() == false && intersects(B) && !open && lastopenedhatch == null){ //Se o personagem intersetar com a placa atribuida ao alcapao, e este estiver fechado e o booleano lastopenedhatch for null
            GreenfootSound pressureplatemusicon = new GreenfootSound("pressureplateon.mp3");
            pressureplatemusicon.play();
            D.setLocation(X+110, Y); //Vai mover o alcapao para a direita
            D.changebeingopen();
            open= true; //muda a variavel open para true
            lastopenedhatch = D;    //e o ultimo alcapao aberto passa para o alcapao dado como atributo
        }
            if(lastopenedhatch == D && open && !intersects(B)){ //Se o ultimo alcapao aberto for o dado, e estiver aberto e, o personagem ja nao esta na placa
            GreenfootSound pressureplatemusicout = new GreenfootSound("pressureplateout.mp3");
            pressureplatemusicout.play();
            D.setLocation(X-110,Y); //move o alcapao para a esquerda
            D.changebeingopen();
            lastopenedhatch = null; // coloca esta variavel a null
            open = false; // e coloca open como false
```

```
        }


    }

    /*Se a vida dos personagens for zero é criada uma variavel auxiliar que guarda o mundo atual.É
       criado um novo mundo em que passa o score obtido no mundo atual para o novo mundo GameOver, seguidamente muda de
    cenário para o mundo criado*/
    public void gameover(){
        if(Health==0){
            UtilWorlds a= (UtilWorlds) getWorld();
            GameOver b= new GameOver((a.getScore()));
            Greenfoot.setWorld(b);
        }

    }
    //Remover um coracao se o persongaem entrar em contact com a classe de damage
    public void health(Class damage, Heart A, Heart B){ //recebe os objetos Heart de cada personagem
        if (timeElapsed == 0){ //Verifica se nao esta em um momento de delay
            if(isTouching(damage)){ //verifica se esta em contacto com algum objeto da classe damage

                    if(Health == 2){ //Se a vida estiver em 2 remove o primeiro coracao e coloca um delay pra que o personagem nao morra instantaneamente
                    GreenfootSound gothit = new GreenfootSound("gothit.mp3");
                    gothit.play();
                    getWorld().removeObject(A); //remove o coracao
                    timeElapsed = timeElapsed+240; //coloca o delay
                }

                if(Health == 1){ //Se estiver a 1 remove o ultimo coracao
                GreenfootSound deadsound = new GreenfootSound("dead.mp3");
                deadsound.play();
                    getWorld().removeObject(B);
                    timeElapsed = timeElapsed+240;
                }
            Health = Health - 1; //diminui a vida em 1 sempre que houve contacto entre o personagem e algum objeto de dano fora do tempo de delay
            }


            }
        else timeElapsed --; //Se estiver em delay vai dimuir o tempo
            }
            }


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

/**
 * Write a description of class Barbie here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class Barbie extends MainCharacters
{
    /*Importa as fotos num array*/
    public GreenfootImage[] AnimationMotion_barbie_right =
    {
        new GreenfootImage("standing_barbie.png"),
        new GreenfootImage("2.png"),
        new GreenfootImage("3.png"),
        new GreenfootImage("4.png")
    };
```

```
    public GreenfootImage[] AnimationMotion_barbie_left =
    {
        new GreenfootImage("standing_barbie.png"),
        new GreenfootImage("2.png"),
        new GreenfootImage("3.png"),
        new GreenfootImage("4.png")
    };

    public Barbie(){
        /*Velocidade,Vida*/
        super(4,2,false,false,false,-30);
        AnimationMotion_barbie_left[0].mirrorHorizontally();
        AnimationMotion_barbie_left[1].mirrorHorizontally();
        AnimationMotion_barbie_left[2].mirrorHorizontally();
        AnimationMotion_barbie_left[3].mirrorHorizontally();
    }

    public void act()
    {
    try{
        CoinColide(CoinBarbie.class);

        MyWorld world = (MyWorld) getWorld();

        movement("D","A","W",AnimationMotion_barbie_right,
        AnimationMotion_barbie_left);
        checkFalling();
        checkCollisionWithGround(Ground.class);
        checkCollisionWithBlockForbidden(BlocksForbidden.class);


        health(damage.class,world.getHeart(2),world.getHeart(3));
        gameover();

        //Abrir portas

        opendoorslevel1();
    }
    catch(Exception e){

        CoinColide(CoinBarbie.class);

        Level2 world = (Level2) getWorld();

        movement("D","A","W",AnimationMotion_barbie_right,
        AnimationMotion_barbie_left);
        checkFalling();
        checkCollisionWithGround(Ground.class);
        checkCollisionWithBlockForbidden(BlocksForbidden.class);


        health(damage.class,world.getHeart(2),world.getHeart(3));
        gameover();
    }

    }
}


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

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

```
 * @version (a version number or a date)
 */
public class Ken extends MainCharacters
{
    public GreenfootImage[] AnimationMotion_ken_right =
    {
        new GreenfootImage("standing_ken.png"),
        new GreenfootImage("2_ken.png"),
        new GreenfootImage("3_ken.png"),
        new GreenfootImage("4_ken.png")
    };
    public GreenfootImage[] AnimationMotion_ken_left =
    {
        new GreenfootImage("standing_ken.png"),
        new GreenfootImage("2_ken.png"),
        new GreenfootImage("3_ken.png"),
        new GreenfootImage("4_ken.png")
    };

    public Ken(){
        /*int Velocity,int Health,boolean FacingRight,boolean Jumping,boolean Walking*/
        super(4,2,true,false,false,-30);
        AnimationMotion_ken_left[0].mirrorHorizontally();
        AnimationMotion_ken_left[1].mirrorHorizontally();
        AnimationMotion_ken_left[2].mirrorHorizontally();
        AnimationMotion_ken_left[3].mirrorHorizontally();
    }
    /**
     * Act - do whatever the ken wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        try{
        CoinColide(CoinKen.class);

        MyWorld world = (MyWorld) getWorld();

        movement("right","left","up",AnimationMotion_ken_right,
        AnimationMotion_ken_left);
        checkFalling();
        checkCollisionWithGround(Ground.class);
        checkCollisionWithBlockForbidden(BlocksForbidden.class);


        health(damage.class,world.getHeart(0),world.getHeart(1));
        gameover();

        //Abrir portas

        opendoorslevel1();


    }
    catch(Exception e){

        CoinColide(CoinKen.class);

        Level2 world = (Level2) getWorld();

        movement("right","left","up",AnimationMotion_ken_right,
        AnimationMotion_ken_left);
        checkFalling();
        checkCollisionWithGround(Ground.class);
        checkCollisionWithBlockForbidden(BlocksForbidden.class);
```

```
        health(damage.class,world.getHeart(0),world.getHeart(1));
        gameover();
    }

    }
}
```

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

/**
 * Write a description of class damage here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class damage extends Actor
{
    /**
     * Act - do whatever the damage wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        // Add your action code here.
    }
}
```

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

/**
 * Write a description of class NavalBomb here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class NavalBomb extends damage
{
    /**
     * Act - do whatever the NavalBomb wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        // Add your action code here.
    }
}
```

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

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

```
     * Act - do whatever the Spikes wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
    public void act()
    {
        // Add your action code here.
    }
}

/**
 * Write a description of class mob here.
 *
 * @author (your name)
 * @version (a version number or a date)
 */
public class mob extends damage
{
    /**
     * Act - do whatever the mob wants to do. This method is called whenever
     * the 'Act' or 'Run' button gets pressed in the environment.
     */
   int speed=-1;
    int count=0;
    public mob()
    {
        getImage().scale(30,30);
    }
    public void monster_movement()
    {
        if(count<100)
        setLocation(getX() +speed,getY());
        else
        {
            speed=-speed;
            getImage().mirrorHorizontally();
            count=0;
        }
    }
    public void act()
    {
        count++;
        monster_movement();
    }

}
```