COMPONENTES:

(CC2A30) GABRIEL ÂNGELO RA: F0940G9

(CC2A30) RENAN RAMOS RA: F0960C9

(CC3A30) JONAS PRUDENCIO RA: N390441

(CC2A30) WILSON PINHEIRO RA: F127HF4

(ATIVIDADE PRÁTICA SUPERVISIONADA – APS)

CIÊNCIA DA COMPUTAÇÃO

(CC2A30 / CC3A30)

Orientador: Prof. Osman Bras

TEMA: “DESENVOLVIMENTO DE UM JOGO COM UTILIZAÇÃO DE INTERFACE GRÁFICA”

NOME DO TRABALHO:

BRASÍLIA 2020

SUMÁRIO

1 - OBJETIVO E MOTIVAÇÃO DO TRABALHO:

O objetivo deste trabalho é sobre o conceito de desenvolvimento sustentável e a preservação do meio ambiente, bem como as suas características, vantagens e desafios a serem superados, buscando expor a sua importância para o futuro da aprendizagem e sua utilização e sua aplicabilidade, por meio de um jogo 2D desenvolvido na linguagem Java.

O projeto deste jogo, não tem somente a finalidade para este propósito, mas também podendo visar futuros projetos de desenvolvimento de jogos e demostrando seu foco de desenvolvimento, e como esta área da computação está atual mente situada no mercado de trabalho atual.

UNIP
UNIVERSIDADE PAULISTA

2 - INTRODUÇÃO

Atualmente, os Jogos Digitais são um dos recursos mais utilizados para realizar a interação entre os estudantes e o objeto de aprendizado. Pois, estes estão no cotidiano dos mesmos como forma de entretenimento e recreação, sendo assim, podem ser utilizados como fator motivacional. Dessa forma, cria-se uma ponte entre atividades lúdicas e conteúdos formais, favorecendo assim, o processo de aprendizagem. Ou seja, são alternativas dinâmicas que propiciam maior interação e diálogo enquanto recurso pedagógico, apoiado em metodologias comumente utilizadas como livros, vídeos, filmes, etc. Sendo assim, o presente trabalho visa realizar uma explanação sobre desenvolvimento sustentável e preservação do meio ambiente com utilização de Jogos Digitais utilizando a linguagem Java criando um jogo na interface gráfica 2D.

Os jogos digitais tem sido referência no mundo há 50 anos, estreado em 1977 o Atari 2600 chegou a vender oito milhões de unidades até 1983. Com a chegada do Atari fez com que o mercado vendesse vários jogos, promovendo ao usuário variedades em sua compra.

Segundo Schuytema, um game é composta por uma série de ações e decisões, que resultam em uma decisão final. A indústria de games não é impulsionada pelos executivos e sim pelos games, assim fica sendo um diferencial no mundo do entretenimento.

Existem games que são criados para um público específico de consumidores, e não para todos. Os desenvolvedores de games analisam o projeto com a perspectiva de alcançar um maior público, também tentam atingir jogadores fantásticos por outros tipos de jogos.

Um novo tipo surgiu na indústria de games, chamado de games casuais. Um game casual é qualquer game que pode ser jogado em um intervalo de tempo, por exemplo os games árcades clássicos. Jogadores casuais são profissionais como médicos, advogados, executivos de negócios, desenvolvedores de software, ou seja, qualquer pessoa.

Esses tipos de games atraem pessoas de todas as culturas, classes sociais, sexo, religião, etnia e orientação política, sendo assim é importante ter a disposição jogos casuais simples, bons, “inocentes”, “puros”, com jogabilidade boa e que divirta o usuário.

Tendo em vista essas considerações, cabe responder a seguinte pergunta: Como é feito um jogo digital?

O presente trabalho busca mostrar para os acadêmicos um jogo funcional, seu plano de desenvolvimento e o desenvolvimento em si. Com pesquisas, avaliações e benefícios, o trabalho busca relatar o processo de desenvolvimento de um jogo digital, desde a sua concepção até sua finalização.

3 - Funcionamento do jogo

O jogo possuirá uma mecânica básica de movimentação horizontal (esquerda e direita) e pulo para cima e um ataque básico com sua espada. Á medida que o jogador completar um mapa, será direcionado ao mapa seguinte. Até chegar ao mapa final onde encontrará um boss que terá de derrotar para finalizar o jogo.

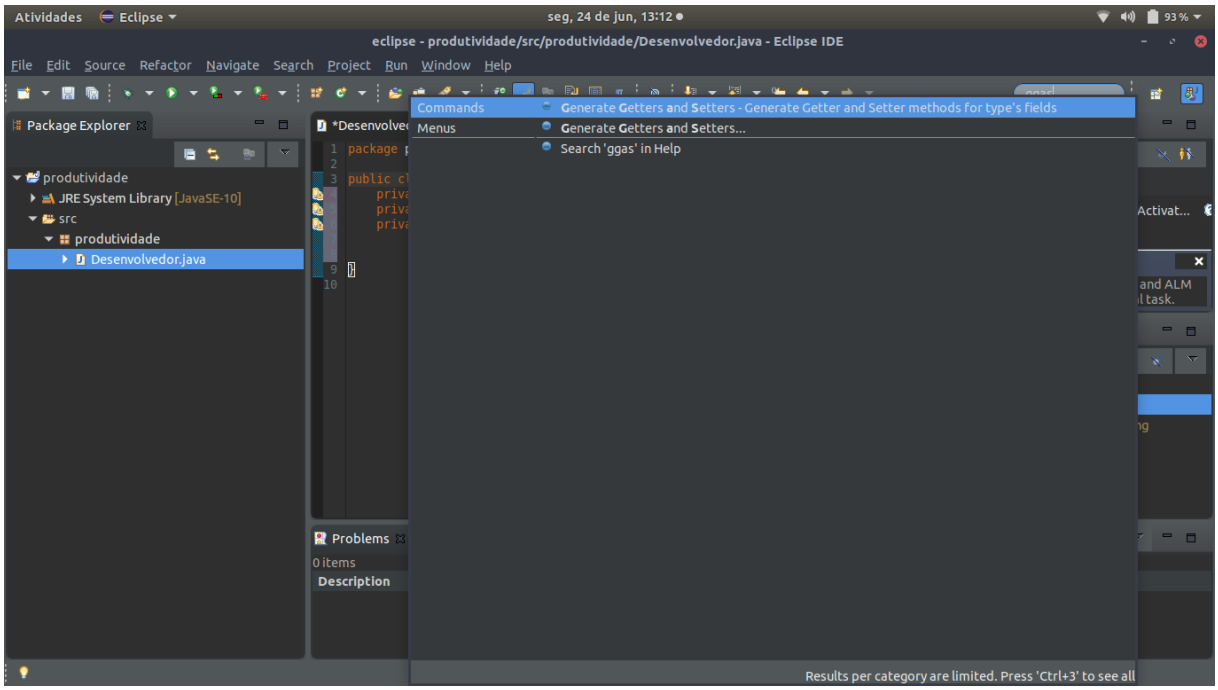
UNIP
UNIVERSIDADE PAULISTA

4 - Plano de desenvolvimento

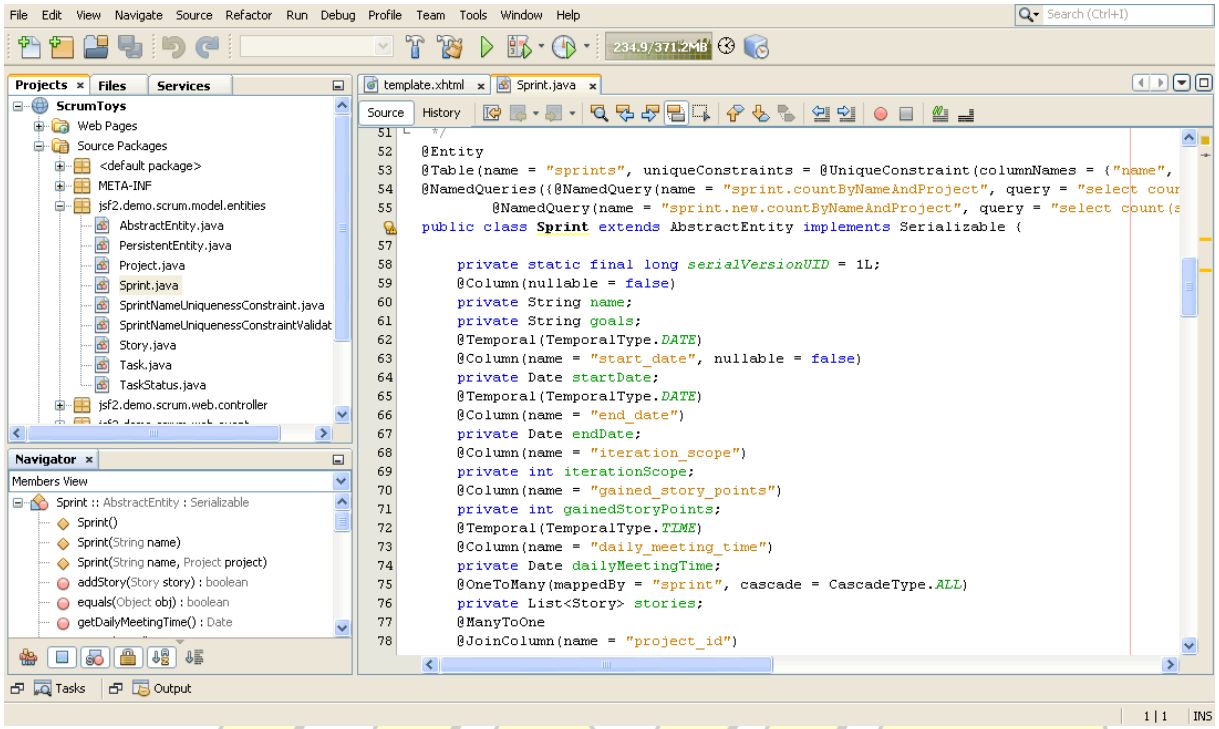
O desenvolvimento será guiado a medida que os mapas e mobs ficarem prontos. Sendo baseado em uma estrutura de models onde cada mapa herdará uma classe geral de mapas e implementará suas peculiaridades. Os elementos utilizados serão pngs dos mapas e elementos internos do mapa e gifs que representarão animações de personagem. Os elementos do jogo vão ser agrupados em suas características similares para facilitar o desenvolvimento com uso de heranças.

De ferramentas serão utilizadas as IDES NetBeans e Eclipse para desenvolvimento do código, Aseprite para criação das pixel arts de personagens e mobs e Photoshop para criação de telas de menu do usuário.

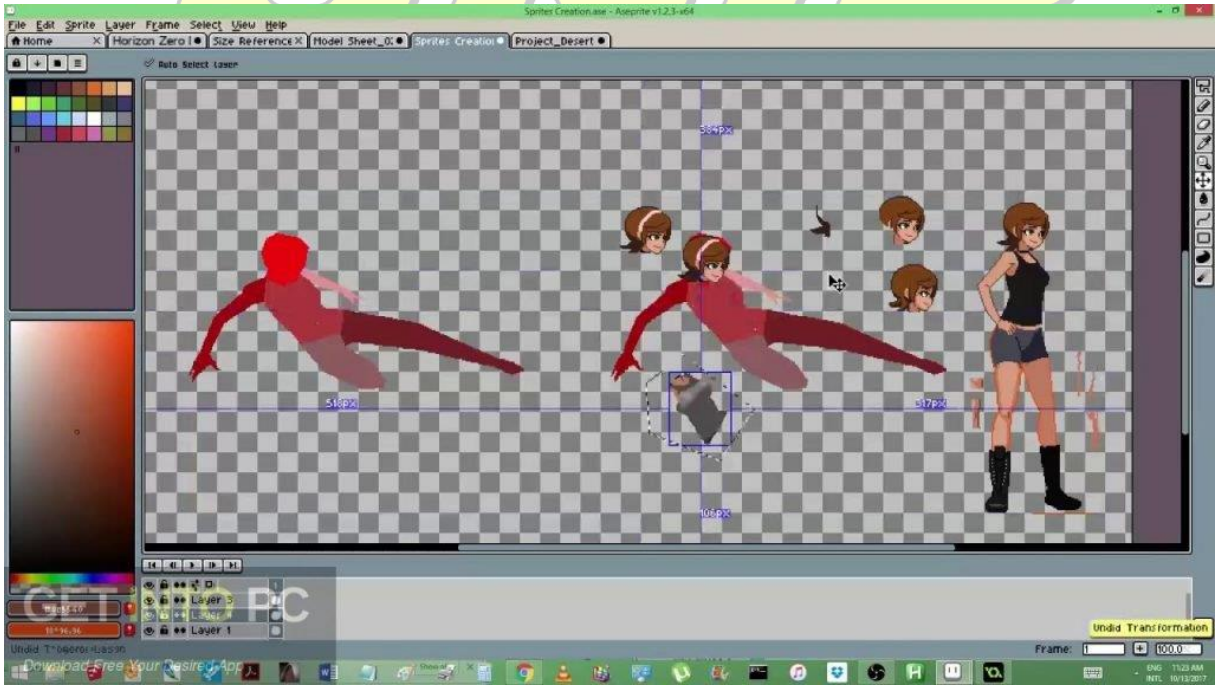
Eclipse:



NetBeans:



Aseprite:



5 - Projeto

O projeto terá as classes principais Game e Game Panel que ficarão com a responsabilidade de configurar a tela que exibirá todo o jogo. A classe GameState será classe modelo para gerenciar os estados do game, como menus, mapas etc. Existirão também classes relacionadas à exibição de mapas de componentes de mapas, tendo como classe modelo a classe Background, entre outras. Além disso possuirá as classes relacionadas ao personagem que definirá mecânicas e animações.

```
package Entity;
import java.awt.image.BufferedImage;
public class Animation {
	private BufferedImage[] frames;
	private int currentFrame;
	private long startTime;
	private long delay;
	private boolean playedOnce;
	public Animation() {
		playedOnce = false;
	}
	public void setFrames(BufferedImage[] frames) {
		this.frames = frames;
		currentFrame = 0;
		startTime = System.nanoTime();
		playedOnce = false;
	}
	public BufferedImage[] getFrames() {
		return this.frames;}
	public void setDelay(long d) {delay = d;
}public void setFrame(int i) {currentFrame = i;}
//determina quando ir para o prox frame
```

```
	public void update( ) {
		if(delay == -1) return;
		long elapsed = (System.nanoTime() - startTime) / 1000000;
		//se o tempo decorrido for menor que o delay
		if(elapsed > delay) {
			currentFrame++;
			startTime = System.nanoTime();}
		if(currentFrame == frames.length ) {
			if(getFrames() == Player.sprites.get(Player.WALKING) ||
getFrames() == Player.sprites.get(Player.IDLE)) {
				currentFrame = 0;
				playedOnce = true;
			}else {
				currentFrame = frames.length - 1;
				playedOnce = true;}}}
	public int getFrame() {return currentFrame;}
	public BufferedImage getImage() {return frames[currentFrame];}
	//retorna se a animação foi executada alguma vez
	public boolean hasPlayedOnce() {return playedOnce;}
}

package Entity;
import TileMap.TileMap;
public class Enemy extends MapObject {
	protected int health;
	protected int maxHealth;
	protected boolean dead;
	protected int damage;
	protected boolean flinching;
	protected long flinchTimer;
	public Enemy(TileMap tm) {
```

```
            super(tm);}
        public boolean isDead() {return dead;}
        public int getDamage() {return damage;}
        public void hit(int damage) {
            if(dead || flinching) return;
            health -= damage;
            if(health < 0) health = 0;
            if(health == 0) { dead = true;
            }
            flinching = true;
            flinchTimer = System.nanoTime();
        }
        public void update() {
        }
}

package Entity;
import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;
public class Explosion {
        private int x;
        private int y;
        private int xmap;
        private int ymap;
        private int width;
        private int height;
        private Animation animation;
        private BufferedImage[]  sprites;
        private boolean remove;
```

```
public Explosion(int x, int y) {
        this.x = x;
        this.y = y;
        width = 120;height = 120;
        try {
BufferedImage spritesheet = ImageIO.read(
                getClass().getResourceAsStream("/mobs/Explosion.png"));
                sprites = new BufferedImage[5];
                for(int i = 0; i < sprites.length; i++) {
                        sprites[i] = spritesheet.getSubimage(
                                        i * width,
                                        0width,height );}
        }catch(Exception e) {
                e.printStackTrace();
        }
        animation = new Animation();
        animation.setFrames(sprites);
        animation.setDelay(70);
}
public void update() {
        animation.update();
        if(animation.hasPlayedOnce()) {
                remove = true;
        }}
public boolean shouldRemove () {return remove;}
public void setMapPosition(int x, int y) {
        xmap = x;
        ymap = y;
}
public void draw(Graphics2D g) {
```

```
                g.drawImage(
                        animation.getImage(),
                        x + xmap - width / 2,
                        y + ymap - height / 2,
                        null);}
}
package Entity;
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;
public class HUD {
        private Player player;
        private BufferedImage image;
        private Font font;
        public HUD(Player p) {
                player = p;
                try {
                        image = ImageIO.read(
                getClass().getResourceAsStream("/HUD/hud.gif));
                        font = new Font("Sans-Serif", Font.BOLD, 25);
                }catch(Exception e) {
                        e.printStackTrace();}
        }
        public void draw(Graphics2D g) {
                g.drawImage(image, 0, 20, 196, 100, null);
                g.setFont(font);
                g.setColor(Color.white);
        g.drawString(player.getHealth() *20 + "/" + (player.getMaxHealth() * 20) ,65,
52);}   }
```

```
package Entity;
import java.awt.Graphics2D;
import java.awt.Rectangle;
import Main.GamePanel;
import TileMap.Tile;
import TileMap.TileMap;
public abstract class MapObject {
	//map stuff
	protected TileMap tileMap;
	protected int tileSize;
	protected double xmap;
	protected double ymap;
	//positions
	protected double x;
	protected double y;
	protected double dx;
	protected double dy;
	//dimensions
	protected int width;
	protected int height;
	//collision box
	protected int cwidth;
	protected int cheight;
	// collision
	protected int currRow;
	protected int currCol;
	protected double xdest;
	protected double ydest;
	protected double xtemp;
```

```
protected double ytemp;
protected boolean topLeft;
protected boolean topRight;
protected boolean bottomLeft;
protected boolean bottomRight;
//animation
protected Animation animation;
protected int currentAction;
protected int previousAction;
protected boolean facingRight;
//movement
protected boolean left;
protected boolean right;
protected boolean up;
protected boolean down;
protected boolean jumping;
protected boolean falling;
protected boolean attacking;
//movement physics
protected double moveSpeed;
protected double maxSpeed;
protected double stopSpeed;
protected double fallSpeed;
//terminal velocity
protected double maxFallSpeed;
protected double jumpStart;
protected double stopJumpSpeed;
//construtor
public MapObject(TileMap tm) {
        tileMap = tm;
```

```
tileSize = tm.getTileSize();}
        public boolean intersects(MapObject o) {
                Rectangle r1 = getRectangle();
                Rectangle r2 = o.getRectangle();
                return r1.intersects(r2);
        }
        public Rectangle getRectangle() {
                return new Rectangle(
                                (int)x - cwidth,
                                (int)y - cheight,
                                cwidth,
                                cheight);
        }
        public void calculateCorners(double x, double y ) {
                int leftTile = (int) (x - cwidth / 2) / tileSize;
                int rightTile = (int) (x + cwidth / 2 - 1) / tileSize;
                int topTile = (int) (y - cheight / 2) / tileSize;
                int bottomTile = (int) (y + cheight / 2 - 1) / tileSize;
                if(topTile < 0 || bottomTile >= (tileMap.getHeight() /
tileMap.getTileSize()) ||
                                leftTile < 0 || rightTile >= (tileMap.getWidth() /
tileMap.getTileSize())){
                                topLeft = topRight = bottomLeft = bottomRight = false;
                                return;
                                }
                //top left
                int tl = tileMap.getType(topTile, leftTile);
                //top right
                int tr = tileMap.getType(topTile, rightTile);
                int bl = tileMap.getType(bottomTile, leftTile);
                int br = tileMap.getType(bottomTile, rightTile);
```

```
            topLeft = tl == Tile.BLOCKED;
            topRight = tr == Tile.BLOCKED;
            bottomLeft = bl == Tile.BLOCKED;
            bottomRight = br == Tile.BLOCKED;
    }
    public void checkTileMapCollision() {
            currCol = (int)x / tileSize;
            currRow = (int)y / tileSize;
            xdest = x + dx;
            ydest = y + dy; xtemp = x; ytemp = y;
            int totalCol = tileMap.getWidth() / tileMap.getTileSize();
            int totalRow = tileMap.getHeight() / tileMap.getTileSize();
            calculateCorners(x, ydest);
            //para cima
            if(dy < 0) {
                    //tocou em algo
                    if(topLeft || topRight) {
                            //para de andar nessa direção
                            //dy = stopSpeed;
                            dy = 0;
                            ytemp = (currRow ) * tileSize + cheight / 2;
                    }else {
                            ytemp += dy;}
            }
            //caindo
            if(dy > 0) {
                    //tocou em algo
                    if(bottomLeft || bottomRight) {
                            dy = 0;
                            falling = false;
```

```
                ytemp = (currRow + 1) * tileSize - cheight/2;
            }else if (currRow == totalRow + 3) {
                System.out.println("to no limbo");
                ytemp = 0;
                xtemp = 0;
            }
            //não tocou em nada
            else {
                ytemp += dy;
            }
        }
        calculateCorners(xdest, y);
        //para esquerda
        if(dx < 0) {
            //tocou na parede
            if(topLeft || bottomLeft){
                dx = 0;
                //seta xtemp para direita do tile
                //xtemp = currCol * tileSize - cwidth / 2;
            }
            /*else if(currCol <=0) {
                dx = 0;
            }*/
            else {
                xtemp += dx;
            }
        }
        //para direita
        if(dx > 0) {
            //tocou na parede
```

```
                if(topRight || bottomRight) {
                        dx = 0;
                        xtemp = (currCol+1) * tileSize - cwidth / 2;
                }/*else if (currCol == totalCol-1) {
                        //xtemp = (totalCol - 1) / tileSize;
                        dx = 0;
                }*/
                else {
                        xtemp += dx;
                }
        }
        if(!falling) {
                calculateCorners(x, ydest + 1);
                //não está em nenhum chão
                if(!bottomLeft && !bottomRight) {
                        falling = true;
                }
        }
    }
    public int getX() {return (int)x; }
    public int getY() {return (int)y; }
    public int getHeight() {return height;}
    public int getWidth() {return width;}
    public int getCWidth() {return cwidth;}
    public int getCHeight() {return cheight;}
    public void setPosition(double x, double y) {
        this.x = x; this.y = y;
    }
void setVector(double dx, double dy) {
        this.dx = dx;
```

```
            this.dy = dy;
        }
        //global position and localposition
        public void setMapPosition() {
            //designa onde o personagem vai ser desenhado
            xmap = tileMap.getX(); ymap = tileMap.getY();
        }
        public void setLeft(boolean b) {left = b;}
        public void setRight(boolean b) {right = b;}
        public void setUp(boolean b) {up = b;}
        public void setDown(boolean b) {down = b;}
        public void setJumping(boolean b) {jumping = b;}
        public boolean notOnScreen() {
            //final position
            return x + xmap + width < 1 || //se o objeto está alem do lado esquedo
        x + xmap - width > GamePanel.WIDTH ||//se está além do lado direito
                                y + ymap + height < 1 || //se está além do topo
    y + ymap - height > GamePanel.height; //se está além do chão}
        public void draw(Graphics2D g) {
            if(facingRight) {
                g.drawImage(
            animation.getImage(),
                (int) (x + xmap - width /2),  (int) (y + ymap - height /2), null);
            }
            else {
                g.drawImage(
                            animation.getImage(),
                            (int)(x + xmap - width / 2 + width),
                            (int)(y + ymap - height / 2),
                            -width, height, null);}}}
```

```
package Entity;
import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import java.util.ArrayList
import javax.imageio.ImageIO;
import TileMap.TileMap;
public class Player extends MapObject {
        private int health;
        private int maxHealth;
        private int power;
        private int maxPower;
        private boolean dead;
        private boolean dodging;
        private long dodgeTime;
        //atacks
        private boolean attacking;
        private int attackCost;
        private int attackDamage;
        private int attackRange;
        protected boolean flinching;
        protected long flinchTimer;
        private boolean scratching;
        private int scratchDamage;
        private int scratchRange;
        //animations
        public static ArrayList<BufferedImage[]> sprites;
        private final int[] numFrames = {
        4, 8, 3, 1, 9};
        public static final int IDLE = 0;
        public static final int WALKING = 1;
```

```
        private static final int JUMPING = 2;
        private static final int FALLING = 3;
        private static final int ATTACK = 4;
        private static final int SCRATCHING = 0;
        public Player(TileMap tm) {
                super(tm);
                // TODO Auto-generated constructor stub
                width = 128; height = 128;cwidth = 50;cheight = 110;moveSpeed = 3;
maxSpeed = 3.4; stopSpeed = 0.85; fallSpeed = 0.32; maxFallSpeed = 8.5;
jumpStart = -14.24; stopJumpSpeed = 0.3; facingRight = true;
health = maxHealth = 5; scratchDamage = 8; scratchRange = 40; attackRange = 20;
attackDamage = 1;
                //load sprites
                try {
                        //carrega sprite
                        BufferedImage spritesheet = ImageIO.read(
                        getClass().getResourceAsStream());
sprites = new ArrayList<BufferedImage[]>();
recorda cada parte do sprite e coloca no vetor de animações
        for(int i = 0; i < 5; i++) {
                        BufferedImage[] bi = new BufferedImage[numFrames[i]];
                for(int j = 0; j < numFrames[i]; j++ ) {
                                if(i == 4) {bi[j] = spritesheet.getSubimage(
                j * 161,i * height,161,height);continue;}
                        bi[j] = spritesheet.getSubimage(
                        j * width,i * height,width,height);}
                                        sprites.add(bi);}
                }catch(Exception e) {
                        e.printStackTrace();
                }
                animation = new Animation();
```

```
            currentAction = IDLE;
            animation.setFrames(sprites.get(IDLE));
            animation.setDelay(150);
    }
    public int getHealth() {return health;}
    public int getMaxHealth() {return maxHealth;}
    public void setScratching() {
            scratching = true;
    }
    public void setAttacking() {
                    attacking = true;}
    public void checkAttack(ArrayList<Enemy> enemies) {
            //loop enemies
            for(int i = 0; i < enemies.size(); i++) {
                    Enemy e = enemies.get(i);
                    if(attacking) {
                            if(facingRight) {
                                    if(
                                                    e.getX() > x &&
                                                    e.getX() < x + attackRange &&
                                                    e.getY() > y - height / 2 &&
                                                    e.getY() < y + height / 2
                                                    ) {
                                            e.hit(attackDamage);}
                            }else {
                                    if(
                                                    e.getX() < x &&
                                                    e.getX() > x - attackRange &&
                                                    e.getY() > y - height / 2 &&
                                                    e.getY() < y + height / 2  ) {
```

```
            e.hit(attackDamage);}  }}
                    //check enemy collision
                    if(intersects(e)) {
                            hit(e.getDamage());
                    }
            }
    }
    public void hit(int damage) {
            if(flinching) return;
            health -= damage;
            if(health < 0) health = 0;
            if(health == 0) dead = true;
            flinching = true;
            flinchTimer = System.nanoTime();
    }
    //verifica tecla apertada e move personagem
    private void getNextPosition() {
            //movement
            if(left) {
                    dx -= moveSpeed;
                    if(dx < -maxSpeed) {
                            dx = -maxSpeed;}}
            else if(right) {
                    dx += moveSpeed;
                    if(dx > maxSpeed) {
                            dx = maxSpeed;}}
            else {
                    if(dx > 0) {
                            dx -= stopSpeed;
                            if(dx < 0) {
```

```
                                dx = 0;
                        }
                }
                else if(dx < 0) {
                        dx += stopSpeed;
                        if(dx > 0) {
dx = 0;         }}}
                //cannot move while attackng, except in air
                if(currentAction == ATTACK) {
                        dx = 0;
                }
                //jumping
                if(jumping && !falling) {
                        dy = jumpStart;
                        falling = true;
                }
                //falling
                if(falling) {
                        dy += fallSpeed;
                        if(dy > 0) jumping = false;
                        if(dy < 0 && !jumping) dy += stopJumpSpeed;
                        if(dy > maxFallSpeed) dy = maxFallSpeed;}
                if(attacking) {
                        if(facingRight) {
                                dx+= 3;
                        }else {
                                dx -= 3;
                        }}}
public void update() {
                //update position
```

```
getNextPosition();
checkTileMapCollision();
setPosition(xtemp, ytemp);
System.out.println(x);
System.out.println(y);
//set animations
if(currentAction == ATTACK) {
                if(animation.hasPlayedOnce()) {
                        attacking = false;
                }       }
 if(attacking) {
         if(currentAction != ATTACK) {
                currentAction = ATTACK;
                        animation.setFrames(sprites.get(ATTACK));
                        animation.setDelay(50);
                        width = 161;
                }
 }
  else if(dy > 0) {
         if(currentAction != FALLING) {
                currentAction = FALLING;
                animation.setFrames(sprites.get(FALLING));
                animation.setDelay(-1);
                width = 128;
         }
}
else if(dy < 0) {
         if(currentAction != JUMPING) {
                currentAction = JUMPING;
                animation.setFrames(sprites.get(JUMPING));
```

```
                animation.setDelay(90);
                width = 128;
            }
        }
        else if(left || right) {
            if(currentAction != WALKING) {
                currentAction = WALKING;
                animation.setFrames(sprites.get(WALKING));
                animation.setDelay(60);
                width = 128;
            }
        }
        else {
            if(currentAction != IDLE) {
                currentAction = IDLE;
                animation.setFrames(sprites.get(IDLE));
                animation.setDelay(150);
                width = 128;
            }
        }
        //flinching
        if(flinching) {
            long elapsed = (System.nanoTime() - flinchTimer) / 1000000;
            if(elapsed > 1000) {
                flinching = false;
            }
        }
        animation.update();
        //set direction
        if(currentAction != ATTACK ) {
```

```
                    if(right) facingRight = true;
                    if(left) facingRight = false;
                }
        }
        public void draw(Graphics2D g) {
                setMapPosition();
                //draw player
        super.draw(g);
        }
}
package GameState;
public abstract class GameState {
        protected GameStateManager gsm;
        public abstract void init();
        public abstract void update();
        public abstract void draw(java.awt.Graphics2D g);
        public abstract void keyPressed(int k);
        public abstract void keyReleased(int k);
}

package GameState;
import java.util.ArrayList;
public class GameStateManager {
        private ArrayList<GameState> gameStates;
        private int currentState;
        public static final int MENUSTATE = 0;
        public static final int LEVEL1STATE = 1;
        public GameStateManager() {
                gameStates = new ArrayList<GameState>();
                currentState = MENUSTATE;
```

```
            gameStates.add(new MenuState(this));
            gameStates.add(new Level1State(this));}
        public void setState(int state) {
            currentState = state;
            gameStates.get(currentState).init();}
        public void update() {
            gameStates.get(currentState).update();
        }
        public void draw(java.awt.Graphics2D g) {
            gameStates.get(currentState).draw(g);}
        public void keyPressed(int k) {
            gameStates.get (currentState).keyPressed(k);
        }
        public void keyReleased(int k) {
            gameStates.get(currentState).keyReleased(k);
        }
}
```



```
package GameState;
import java.awt.Color;
import java.awt.Graphics2D;
import java.awt.Point;
import java.awt.event.KeyEvent;
import java.io.File;
import java.util.ArrayList;
import Entity.Enemy;
import Entity.Explosion;
import Entity.HUD;
```

```
import Entity.Player;
import Entity.Enemies.Pet;
import Main.GamePanel;
import TileMap.Background;
import TileMap.TileMap;
import handlers.Keys;
public class Level1State extends GameState {
	private TileMap tilemap;
	private Background bg;
	private Background bg2;
	private Player player;
	private ArrayList<Enemy> enemies;
	private ArrayList<Explosion> explosions;
	private HUD hud;
	public Level1State(GameStateManager gsm) {
		this.gsm = gsm;
		init();  }
	public void init() {
		tilemap = new TileMap(64);
		tilemap.loadTiles("/Tilesets/tilehomes.png");
		tilemap.loadMap("/Maps/hehe.map");
		tilemap.setPosition(0, 0);
		tilemap.setBounds(tilemap.getWidth() - 1 * tilemap.getTileSize(),
		tilemap.getHeight() - 2 * tilemap.getTileSize(),0, 0);
		tilemap.setTween(0.07);
			bg = new Background("/assets/SKY.png", 0.1);
			bg2 = new Background("/assets/MATO.png", 0.2);
			player = new Player(tilemap);
			player.setPosition(150, 460);
		populateEnemies();
```

```
        explosions = new ArrayList<Explosion>();
        hud = new HUD(player);    }
private void populateEnemies() {
        enemies = new ArrayList<Enemy>();
        Pet p;
        Point[] points = new Point[] {
                new Point(1946, 585),
                new Point(2777, 457),
                new Point(5926, 521),
                new Point(7833, 393),
                new Point(8473, 393),
                new Point(12891, 585),
                new Point(13954, 585),
                new Point(14170, 585),
                new Point(14873, 585),
        };
        for(int i = 0; i < points.length; i++ ) {
                p = new Pet(tilemap);
                p.setPosition(points[i].getX(), points[i].getY());
                enemies.add(p);}    ;}
public void update() {
        handleInput();
        player.update();
        //mountains.setPosition(tileMap.getx(), tileMap.gety());
        tilemap.setPosition(
                        GamePanel.WIDTH / 2 - player.getX(),
                        GamePanel.height / 2 - player.getY()
                        );
        //set background
        bg2.setPosition(tilemap.getX(), tilemap.getY());
```

```
		//attack enemies
		player.checkAttack(enemies);
		//update all enemies
		for(int i = 0; i < enemies.size(); i++) {
			Enemy e = enemies.get(i);
			e.update();
			if(e.isDead()) {
				enemies.remove(i);
				i--;
				explosions.add(
					new Explosion(e.getX(), e.getY()));}		}
		//update explisions
		for(int i = 0; i < explosions.size(); i++) {
			explosions.get(i).update();
			if(explosions.get(i).shouldRemove()) {
				explosions.remove(i);
				i--;}}}
	public void draw(Graphics2D g) {
		bg.draw(g);
		bg2.draw(g);
		//draw tilemap
		tilemap.draw(g);
		//draw player
		player.draw(g);
		//draw enemies
		for(int i = 0; i < enemies.size(); i++) {
			enemies.get(i).draw(g);}
		//draw explosions
		for(int i = 0; i < explosions.size(); i++) {
			explosions.get(i).setMapPosition((int)tilemap.getX(),
(int)tilemap.getY());
```

```
                explosions.get(i).draw(g);
            }
            //draw hud
            hud.draw(g); }
public void handleInput() {
        }
        public void keyPressed(int k) {
            if(k == KeyEvent.VK_LEFT) player.setLeft(true);
            if(k == KeyEvent.VK_RIGHT) player.setRight(true);
            if(k == KeyEvent.VK_UP) player.setUp(true);
            if(k == KeyEvent.VK_DOWN) player.setDown(true);
            if(k == KeyEvent.VK_SPACE || k == KeyEvent.VK_UP)
player.setJumping(true);
            if(k == KeyEvent.VK_R) player.setAttacking(); }
        public void keyReleased(int k) {
            if(k == KeyEvent.VK_LEFT) player.setLeft(false);
            if(k == KeyEvent.VK_RIGHT) player.setRight(false);
            if(k == KeyEvent.VK_UP) player.setUp(false);
            if(k == KeyEvent.VK_DOWN) player.setDown(false);
            if(k == KeyEvent.VK_SPACE || k == KeyEvent.VK_UP)
player.setJumping(false);
        //      if(k == KeyEvent.VK_R) player.setAttacking(false); }}


package GameState;
import java.awt.Color;
import java.awt.Font;
import java.awt.Graphics2D;
import java.awt.Polygon;
import java.awt.event.KeyEvent;
import javax.swing.JComboBox.KeySelectionManager;
import TileMap.Background;
```

```
import handlers.Keys;
public class MenuState extends GameState {
	private Background bg;
	private int currentChoice = 0;
	private String[] options = {
			"Start",
			"Help",
			"Quit"
	};
	private Color titleColor;
	private Font titleFont;
	private Font font;
	public MenuState(GameStateManager gsm) {
	this.gsm = gsm;
	try {
		bg = new Background("/assets/backhome.png", 500);
		bg.setVector(-0.8, 0);
		titleColor = new Color(	43, 36, 34);
		titleFont = new Font("Sans Serif", Font.PLAIN, 78);
		font = new Font("Arial", Font.PLAIN, 28);
	}catch(Exception e) {
		e.printStackTrace();}}
	@Override
	public void init() {
		// TODO Auto-generated method stub
	}
	@Override
	public void update() {
		// TODO Auto-generated method stub
		bg.update();
```

```
            //handleInput();
        }
        @Override
        public void draw(Graphics2D g) {
            // TODO Auto-generated method stub
            bg.draw(g);
            //draw title
            g.setColor(titleColor);
            g.setFont(titleFont);
            g.drawString("Provirоment", 440, 240);
            //draw menu option
            g.setFont(font);
            for(int i = 0; i < options.length; i++) {
                if(i == currentChoice) {
                    g.setColor(new Color(173, 17, 55));
                    g.fillRect(565, 323 + i * 45, 158, 35);
                    g.setColor(new Color(94, 16, 35));
                }else {
                    g.clearRect(0, 0, 0, 0);
g.setColor(new Color(173, 17, 55))}g.drawString(options[i], 615, 350 + i * 45);}}
        public void select() {
            if(currentChoice == 0) {
                gsm.setState(GameStateManager.LEVEL1STATE);
            }
            if(currentChoice == 1) {
            }
            if(currentChoice == 2) {
                System.exit(0);
            }
        }
```

```
	@Override
	public void keyPressed(int k) {
		// TODO Auto-generated method stub
		if(k == KeyEvent.VK_ENTER) {
			select();
		}
		if(k == KeyEvent.VK_UP) {
			currentChoice--;
			if(currentChoice == -1) {
				currentChoice = options.length - 1;}
		}
		if(k == KeyEvent.VK_DOWN) {
			currentChoice++;
			if(currentChoice == options.length) {
				currentChoice = 0;}
		}
	}
@Override
	public void keyReleased(int k) {
	// TODO Auto-generated method stub}}
package handlers;
import java.awt.event.KeyEvent;
public class Keys {
	public static final int NUM_KEYS = 16;
	public static boolean keyState[] = new boolean[NUM_KEYS];
	public static boolean prevKeyState[] = new boolean[NUM_KEYS];
	public static int UP = 0;
	public static int LEFT = 1;
	public static int DOWN = 2;
	public static int RIGHT = 3;
	public static int BUTTON1 = 4;
```

```
	public static int BUTTON2 = 5;
	public static int BUTTON3 = 6;
	public static int BUTTON4 = 7;
	public static int ENTER = 8;
	public static int ESCAPE = 9;
	public static int SPACE = 10;
	public static void keySet(int i, boolean b) {
		if(i == KeyEvent.VK_UP) keyState[UP] = b;
		else if(i == KeyEvent.VK_LEFT) keyState[LEFT] = b;
		else if(i == KeyEvent.VK_DOWN) keyState[DOWN] = b;
		else if(i == KeyEvent.VK_RIGHT) keyState[RIGHT] = b;
		else if(i == KeyEvent.VK_W) keyState[BUTTON1] = b;
		else if(i == KeyEvent.VK_E) keyState[BUTTON2] = b;
		else if(i == KeyEvent.VK_R) keyState[BUTTON3] = b;
		else if(i == KeyEvent.VK_F) keyState[BUTTON4] = b;
		else if(i == KeyEvent.VK_ENTER) keyState[ENTER] = b;
		else if(i == KeyEvent.VK_ESCAPE) keyState[ESCAPE] = b;
		else if(i == KeyEvent.VK_SPACE) keyState[SPACE] = b;
	}	public static void update() {
for(int i = 0; i < NUM_KEYS; i++) {
		prevKeyState[i] = keyState[i];}}
	public static boolean isPressed(int i) {
		return keyState[i] && !prevKeyState[i];}
	public static boolean anyKeyPress() {
		for(int i = 0; i < NUM_KEYS; i++) {
			if(keyState[i]) return true;
		}
		return false;}}
package Main;
import javax.swing.JFrame;
```

```
public class Game {
	public static void main(String[] args) {
		JFrame window = new JFrame("Proviroment");
		window.setContentPane(new GamePanel());
		window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
		window.setResizable(false);
		window.pack();
		window.setVisible(true);}}

package Main;
import java.awt.Dimension;
import java.awt.Graphics;
import java.awt.Graphics2D;
import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;
import java.awt.image.BufferedImage;
import javax.swing.JPanel;
import GameState.GameStateManager;
import handlers.Keys;
public class GamePanel extends JPanel implements Runnable, KeyListener {
	public static final int WIDTH = 1280;
	public static final int height = 708;
	public static final int SCALE = 1;
	//game thread
	private Thread thread;
	private boolean running;
	private int FPS = 60;
	private long targetTime = 1000/ FPS;
	//image
	private BufferedImage image;
```

```
private Graphics2D g;
//game state manager
private GameStateManager gsm;
public GamePanel() {
        super();
        setPreferredSize(
                        new Dimension(WIDTH * SCALE, height * SCALE));
                        setFocusable(true);
                        requestFocus();}
public void addNotify() {super.addNotify();
        if(thread == null) {thread = new Thread(this);
                addKeyListener(this);
                thread.start();}}
private void init() {
        image = new BufferedImage(
                        WIDTH, height,
                        BufferedImage.TYPE_INT_RGB);
        g = (Graphics2D) image.getGraphics();

        running = true;
        gsm = new GameStateManager();}
public void run() {
        init();
        long start;
        long elapsed;
        long wait;
        //game loop
        while(running) {
                start = System.nanoTime();
                update();
```

```
                draw();
                drawToScreen();
                elapsed = System.nanoTime() - start;
                wait = targetTime - elapsed / 1000000;
                if(wait < 0) wait = 5;
                try {
                        Thread.sleep(wait);
                }catch(Exception e ) {
                        e.printStackTrace();
                }}}
private void update() {
        gsm.update();
        Keys.update();
}
private void draw() {
        gsm.draw(g);}
private void drawToScreen() {
        Graphics g2 = getGraphics();
        g2.drawImage(image, 0, 0, null);
        g2.dispose();}
public void keyTyped(KeyEvent key) {
}
public void keyPressed(KeyEvent key) {
        gsm.keyPressed(key.getKeyCode());
        //Keys.keySet(key.getKeyCode(), true);
}
public void keyReleased(KeyEvent key) {
        gsm.keyReleased(key.getKeyCode());
        //Keys.keySet(key.getKeyCode(), false);}        }
```

```
package TileMap;
import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import javax.imageio.ImageIO;
import Main.GamePanel;
public class Background {
        private BufferedImage image;
        private double x;
        private double y;
        private double dx;
        private double dy;
        private double moveScale;
        public Background(String s, double ms) {
                System.out.println(s);
                try {
                        image = ImageIO.read(
                                        getClass().getResourceAsStream(s));
                        moveScale = ms;
                        }catch(Exception e) {
                        e.printStackTrace();}         }
        public void setPosition(double x, double y) {
                this.x = (x * moveScale) % GamePanel.WIDTH;
                this.y = (y * moveScale) % GamePanel.height;}
        public void setVector(double dx, double dy) {
                this.dx = dx;
                this.dy = dy;  }
        public void update() {
                x += dx;
                y += dy;}
        public void draw(Graphics2D g) {
```

```
                if(g == null) {
                        System.out.println("IMAGE NULL");
                        System.exit(0);}
                g.drawImage(image, (int)x, (int)y, null);
                if(x < 0) {
                g.drawImage(image,(int)x + GamePanel.WIDTH,
                                (int)y, null);}
        if(x > 0) {
                g.drawImage(image,(int)x - GamePanel.WIDTH,
                                (int)y,null);}}}
package TileMap;
import java.awt.image.BufferedImage;
public class Tile {
        private BufferedImage image;
        private int type;
//tile types
        public static final int NORMAL = 0;
        public static final int BLOCKED = 1;
        public Tile(BufferedImage image, int type) {
                this.image = image;
                this.type = type;}
        public BufferedImage getImage() {
                return image;}
        public int getType() {return type;}}

package TileMap;
import java.awt.Graphics2D;
import java.awt.image.BufferedImage;
import java.io.BufferedReader;
import java.io.File;
```

```
import java.io.InputStream;
import java.io.InputStreamReader;
import javax.imageio.ImageIO;
import Main.GamePanel;
public class TileMap {
	//posições
	private double x;
	private double y;
	//limites
	private int xmin;
	private int ymin;
	private int xmax;
	private int ymax;
	private double tween;
	//mapa
	private int [][] map;
	private int tileSize;
	private int numRows;
	private int numCols;
	private int width;
	private int height;
	//tileset
	private BufferedImage tileset;
	private int numTilesAcross;
	private Tile[][] tiles;
	//drawing
	private int rowOffset;
	private int colOffset;
	private int numRowsToDraw;
	private int numColsToDraw;
```

```
public TileMap(int tileSize) {
        this.tileSize = tileSize;
        numRowsToDraw = GamePanel.height / tileSize +2;
        numColsToDraw = GamePanel.WIDTH / tileSize + 2;
        tween = 1;}
public void loadTiles(String s) {
        try {tileset = ImageIO.read(
                        getClass().getResourceAsStream(s));
            //numero de blocos no tile
            numTilesAcross = tileset.getWidth() / tileSize;
            tiles = new Tile[2][numTilesAcross];
            BufferedImage subimage;
            for(int col = 0; col < numTilesAcross; col++) {
                //subimage recorta uma parte da imagem
                subimage = tileset.getSubimage(
                        col * tileSize,0,tileSize,tileSize);
                tiles[0][col] = new Tile(subimage, Tile.NORMAL);
                subimage = tileset.getSubimage(
                        col * tileSize,
tileSize, tileSize, tileSize);
                tiles[1][col] = new Tile(subimage, Tile.BLOCKED); }
        }catch(Exception e) {
            e.printStackTrace();
        }
    }
    public void loadMap(String s) {
        try {
            InputStream in = getClass().getResourceAsStream(s);
            BufferedReader br = new BufferedReader(
                    new InputStreamReader(in));
```

```
                numCols = Integer.parseInt(br.readLine());
                numRows = Integer.parseInt(br.readLine());
                map = new int[numRows][numCols];
                width = numCols * tileSize;
                height = numRows * tileSize;
                xmin = GamePanel.WIDTH - width;
                xmax = 0;
                ymin = GamePanel.height - height;
                System.out.println("yMIN"+ymin);
                ymax = 0;
                String delims = "\\s+";
                for(int row = 0; row < numRows; row++) {
                        String line = br.readLine();
                        String[] tokens = line.split(delims);
                        for(int col = 0; col < numCols; col++) {
                                map[row][col] = Integer.parseInt(tokens[col]);}}
        }catch(Exception e) {
                e.printStackTrace();}}
public int getTileSize() {return tileSize;}
public double getX() {return x;}
public double getY() {return y;}
public int getWidth() {return width;}
public int getHeight() {return height;}
public int getType(int row, int col) {
        int rc = map[row][col];
        int r = rc / numTilesAcross;
        int c = rc % numTilesAcross;
        return tiles[r][c].getType();}
public void setPosition(double x, double y) {
        //suaviza movimento de camera
```

```
        this.x += (x - this.x) * tween;
        this.y += (y - this.y) * tween;
        fixBounds();
        //coluna por onde vai começar a desenhar
        colOffset = (int)-this.x / tileSize;
        //linha por onde vai começar a desenhar
        rowOffset = (int)-this.y  / tileSize;}
public void setBounds(int i1, int i2, int i3, int i4) {
        xmin = GamePanel.WIDTH - i1;
        ymin = GamePanel.WIDTH - i2;
        xmax = i3;ymax = i4;}
public void setTween(double tween) {
        this.tween = tween;}
//não deixa player sair da tela
private void fixBounds() {
        if(x < xmin) x = xmin;
        if(y < ymin) y = ymin;
        if(x > xmax) x = xmax ;
        if(y > ymax) y = ymax;}
public void draw(Graphics2D g) {
        if(x < xmin) x = xmin;
        for(int row = rowOffset; row < rowOffset + numRowsToDraw; row++) {
                if(row >= numRows) break;
        for(int col = colOffset; col < colOffset + numColsToDraw; col++) {
                        if(col >= numCols) break;
                        if(map[row][col] == 0) continue;
                        int rc = map[row][col];
                        int r = rc / numTilesAcross;
                        int c = rc % numTilesAcross;
                g.drawImage(
```

```
tiles[r][c].getImage(),
(int)x + col * tileSize,
(int)y  + row * tileSize,
null);   }}}}
```

UNIP
UNIVERSIDADE PAULISTA

6 - CONCLUSÃO

Neste trabalho exploramos o conceito de jogo em 2D sob o desenvolvimento sustentável e preservação do meio ambiente. Inicialmente, foi apresentada uma conceituação mais abstrata seguida por um resumo dos principais aspectos presentes na teria proposta no trabalho.

Os responsáveis pela tomada de decisões em editoras de games escolhem os projetos com maior alcance de público possível, e no nosso trabalho não apenas tentamos alcançar o maior número de pessoas, como buscamos deixar claro o objetivo do jogo, para que o jogador se sinta realmente dentro do game.

Alguns dos pilares para a construção de um jogo são: programação, modelagem, sonoplastia, roteirismo, plano de negócios e teste.

Na parte de programação foi necessário algumas visitas a sites e também tiramos dúvidas com profissionais da área, de criação de jogos 2D. Foram encontradas algumas dificuldades, como por exemplo, a precariedade de conteúdo em qualquer língua, mesmo sendo utilizada uma das linguagens de programação mais conhecida do mundo, não foi encontrado material de apoio. O paradigma orientado a objetos, foi um facilitador para o desenvolvimento do game, pois possibilitou uma abstração maior do mundo real e também conseguimos desenvolver o que o professor ensinou na aula, que foi fundamental.

A modelagem, assim como a programação, é um dos pilares de um desenvolvimento de jogo, pois é ela que cuida da parte gráfica do game, possibilitando um maior nível de entretenimento do game. Os programas de edições de imagens (Photoshop e Gimp) foram fundamentais para a qualidade visual do game.

O roteiro do jogo foi inspirado na situação atual do planeta Terra, onde o mesmo se encontra com sérios problemas ambientais, causados pelo próprio ser humano, a poluição. Além da situação do planeta Terra, buscamos inspiração na ignorância humana, da qual tende de aumentar a cada geração.

O som ele aumenta a capacidade de realidade de um jogo, não importa qual, e também o deixa mais empolgante e prazeroso. Esse é um dos motivos que faz com que a sonoplastia seja um dos pilares.

Concluímos que a criação de jogo não envolve apenas a parte de programação e que ela é uma das, senão a, última a ser iniciada..

7 - REFERÊNCIAS BIBLIOGRÁFICAS:

1 - **Desenvolvendo uma aplicação passo a passo**
Disponível em : <https://www.devmedia.com.br/desenvolvendo-uma-aplicacao-passo-a-passo/27337>. Acesso em: 21 abr. 2020

2 - **Importância dos jogos computadorizados na educação**
Disponível em: <https://www.iped.com.br/materias/educacao-e-pedagogia/importancia-jogos-computadorizados-educacao.html>. Acesso em: 21 abr. 2020

3 - **APRENDIZAGEM POR MEIO DE JOGOS DIGITAIS: UM ESTUDO DE CASO DO JOGO ANIMAL CROSSING** Disponível em: <http://www.opet.com.br/faculdade/revista-pedagogia/pdf/n8/artigo-4.pdf>. Acesso em: 23 abr. 2020

4 - **Interfaces gráficas com Swing** Disponível em: <https://www.caelum.com.br/apostila-java-testes-xml-design-patterns/interfaces-graficas-com-swing/> Acesso em 23 abr. 2020.

5 **- Introdução a Interface GUI no Java** Disponível em: < https://www.devmedia.com.br/introducao-a-interface-gui-no-java/25646> Acesso em 24 abr. 2020

6- **Poluição** Disponível em: < https://www.infoescola.com/ecologia/poluicao/> Acesso em 26 abr. 2020