**Changed the clear method from default to public.**

**package** sonar.gamestates.states.levels;

**import** sonar.gamestates.states.levels.stages.entities.Entity;

**import** sonar.gamestates.states.levels.stages.entities.animations.tiles.Tile;

**public** **class** Screen

{

**private** **int** width, height, xOffset, yOffset;

**private** **int**[] pixels;

**public** Screen(**int** width, **int** height)

{

//Renders any entity to the window in the game.

**this**.width = width;

**this**.height = height;

pixels = **new** **int**[width \* height];

}

**public** **void** clear()

{

**for**(**int** i = 0; i < pixels.length; i++){pixels[i] = 0;}

}

**public** **void** renderTile(Tile tile)

{

render(tile, tile.getX(), tile.getY(), tile.getWidth(), tile.getHeight());

}

**private** **void** render(Entity entity, **int** xp, **int** yp, **int** eWidth, **int** eHeight)

{

xp -= xOffset;

yp -= yOffset;

**int** hiddenColour = 0;

**for**(**int** y = 0; y < eHeight; y++)

{

**int** ya = y + yp;

**for**(**int** x = 0; x < eWidth; x++)

{

**int** xa = x + xp;

**if**((xa < 0 || xa >= width) || (ya < 0 || ya >= height)) **break**;

**int** colour = 0;

**int** colourSize = 0;

**int**[] colours = **null**;

**if**(entity.getType() == "Tile")

{

Tile tile = (Tile) entity;

colour = tile.getSprite().getPixels()[x + y \* eWidth];

hiddenColour = tile.getSprite().getHiddenColour();

colourSize = tile.getSprite().getColourSize();

colours = tile.getSprite().getColours();

}

**if**(colour != hiddenColour)

{

**if**(colourSize >= 2 && colourSize < 257){colour = changeColour(colour, colourSize, colours);}

pixels[xa + ya \* width] = colour;

}

}

}

}

**private** **int** changeColour(**int** colourRead, **int** colourSize, **int**[] colours)

{

//A method that allows us to utilize grayscale colour sprites.

**if**(colourRead != 0xffffffff)

{

**int** index = 1;

**if**(colourSize >= 3 && colourSize < 7)

{

**switch**(colourSize)

{

**case** 3:

colourRead = colours[index];

**break**;

**case** 4:

**if**(colourRead == 0xff555555) colourRead = colours[1];

**if**(colourRead == 0xffaaaaaa) colourRead = colours[2];

**break**;

**case** 5:

**if**(colourRead == 0xff404040) colourRead = colours[1];

**if**(colourRead == 0xff808080) colourRead = colours[2];

**if**(colourRead == 0xffc0c0c0) colourRead = colours[3];

**break**;

**case** 6:

**if**(colourRead == 0xff333333) colourRead = colours[1];

**if**(colourRead == 0xff666666) colourRead = colours[2];

**if**(colourRead == 0xff999999) colourRead = colours[3];

**if**(colourRead == 0xffcccccc) colourRead = colours[4];

**break**;

**default**:

System.***out***.println("Something went wrong!");

}

}

**else**

{

**int** alpha = 0xff000000;

**int** colourMin = getColourIncrement(colourSize);

**int** colourMax = 0x00ffffff;

**int** colourInc = colourMin;

**for**(**int** currentColour = colourMin; currentColour < colourMax; currentColour += colourInc)

{

**int** cur = currentColour + alpha;

**if**(cur == colourRead)

{

colourRead = colours[index];

**break**;

}

index++;

}

}

}

**else**

{

colourRead = colours[colourSize - 1];

}

**return** colourRead;

}

**private** **int** getColourIncrement(**int** colourSize)

{

**int** increment = 0;

**switch**(colourSize)

{

**case** 7:

increment = 0x002b2b2b;

**break**;

**case** 8:

increment = 0x00242424;

**break**;

**case** 9:

increment = 0x00202020;

**break**;

**case** 10:

increment = 0x001c1c1c;

**break**;

**case** 11:

increment = 0x001a1a1a;

**break**;

**case** 12:

increment = 0x00171717;

**break**;

**case** 13:

increment = 0x00151515;

**break**;

**case** 14:

increment = 0x00141414;

**break**;

**case** 15:

increment = 0x00121212;

**break**;

}

**return** increment;

}

**public** **void** setOffset(**int** xOffset, **int** yOffset)

{

**this**.xOffset = xOffset;

**this**.yOffset = yOffset;

}

**public** **int**[] getPixels(){**return** pixels;}

**public** **int** getWidth(){**return** width;}

**public** **int** getHeight(){**return** height;}

}