Namless_King

2D Game from Scratch

About the Game

The Nameless King is a 2D Game created with JAVA Swing. As IDE i used IntelliJ from JetBrains. The project took me around 1/2 week to plan and around 4 weeks to program (without writing documentation). The enemies and the player images are "stolen" and inspired by the game Dead Cells from Motion Twin (a company from France) and Evil Empires (a Team of ex-Motion Twin developers and Graphic Designers but also new Recruits).

I did not come as far as I wanted. I wanted to make it possible to move the player and fight against enemies (invent an Inventory). But it took a lot of time to create a level editor (that makes it possible to drag and drop entities, get the coordinates and create a function that makes it possible to change the grid size), but more on that later.

Dead Cells inspired me. I wanted to try and make my own version of the game. Dead cells were made with the Framework "Haxe and Heaps" but I don't have the resources and not the motivation to learn a new language (and I don't want to steal the game completely).

The idea

The idea was to create a simple 2D point and click game (as I mentioned before) I created a simple BluePrint how the levels should look like:

▶ Click to expand the BluePrint

added a Constructor and displayed an NPC as an Image. I created a class "Assets" that holds all the information I need to display it on the Window I created. The class looks like this one here (later I added more Features and Functions. I also began to add Items but I did not finish them yet. More on that later):

▶ Click here to expand

Think

After that, I thought I could add some "Thoughts" to the Player.
Here is the function:

```
public static String think(Main m){
            int i = (int) Math.round(Math.random() * (thoughts.length - 1));
            if (i == m.lastThought){
                return think(m);
            }
            m.lastThought = i;
            return thoughts[i];

        }
```

This function creates a random number and returns a String at the position [i] from the thoughts-array.

```
public static String[] thoughts = {
            "Imagine using Java, kinda cringe",
            "Why are we still here? Just to suffer?",
            "*thinking intensifies*",
            "Sometimes my genius is... It's almost frightening"
        };
```

The function 'think()' gets called in the NPC class from a fuction called say()

▶ Click here to see the function

I'll explain this function. First I check if anything is written there. If it is, I clear it. Then I add a new JLable to the screen, make it visible, and add a white foreground (so that the text is written appears white). After that, I set the Bounds and the Font I want to use. The first for-loop you see is for displayed text gets removed first before another text is written otherwise, the new text gets smashed into the old one. The second for-loop is the "animation" that looks written and doesn't get displayed as a whole thing. After that, the Thought gets canceled after 50 milliseconds. Last but not least, I make the function executable in the run()-function.

Resize

The resize fuction in NPC is the most complex, so I will explain this function (the other resize() function working similar). First we calculate the new xMultiplier and the yMultiplier. If the Multipliers are below 1 and above 0 than the window gets smaller. If the Multiplier is bigger than 1, then the Window gets bigger. We get the xMuitplier by dividing the scene-width through the Assets-window-width 1936 * 882. We get the yMultiplier by subtracting the dialogsection.height from the scene-height and dividing it by Assets-heigth.

`JAVA double xMultiplier = scene.window.getWidth() / Assets.refrenceRect.getWidth(); double yMultiplier =`

`(scene.window.getRootPane().getHeight() - Assets.Scalings.get("dialog-section")) / Assets.refrenceRect.getHeight();` After that happend the function calculates the new x and new y position of the Window and the new width and new height by multipling the old position by the xMultiplier and yMultiplier and the new width and new height, also by multipling the old width and old height by the xMultiplier and yMultiplier.

````JAVA int newX = (int) (originalRect.getX() * xMultiplier); int newY = (int) (originalRect.getY() * yMultiplier);

```
 int newW               = (int) (originalRect.getWidth()  * yMultiplier);
int newH               = (int) (originalRect.getHeight() * yMultiplier);

Rectangle relativeRect = new Rectangle(newX, newY, newW, newH);
setBounds(relativeRect);
```

```` If the NPC speaks at that moment, it gets also resized. First we check if the NPC speeks. If he does, basicly the same proceder started over again and resizes the speech with a simillar method as the NPC gets resized.

# Level Editor

The LevelEditor is a class that makes it possible to drag and drop enteties on the window and get the coordinates from the object moved around. There are a few functions, most of them selfexplaining.

- mouseDragged(MouseEvent e), changes the value of the entity.
- mouseReleased(MouseEvent e), calculates the new Position of the entity and prints it on the console.
- startDragging(Component c, int x, int y), tells where the Image starts getting dragged and dropped.
- LevelEditor(Window win), activates on the given Window and prints a simple message in the console.

▶ Click to expand the LevelEditor

# Level

```mermaid

graph TD;

JFrame --> Window; Scene --> Window; Window --> Scene; Scene --> SomeLevel;

```
 This is a simplyfied version of a Level. A Level extends Scene. In Scene is an Object from Window and in Window is an
Object of Scene. Window extends JFrame (from JAVA Swing).

## Window Class
In the Window class I created a function setScene(Scene scene)
that helps me to load Scenes into my Window (more on that later). It also checks if the Key ESCAPE is pressed and if, the I close
the game-window.

````JAVA
    public void setScene(Scene scene){
        this.scene = scene;
        scene.load();
        this.revalidate();

        this.addKeyListener(new KeyAdapter() {
            @Override
            public void keyPressed(KeyEvent e) {
                if(e.getKeyCode() == KeyEvent.VK_ESCAPE) {
                    System.exit(0);
                }
            }
        });
    }
```

In the Constructor from the Window-class we set a title for the game-window.

- make the window Fullscreen
- load the NPC
- give the window a width and a height
- set it visible
- define a minimum size
- add a Background
- and set the Layout of the ContentPane to null

After that I added a ComponentListener that tells me if the window gets resized. I also check (if the level-editor is used) if an object is moved.

▶ Click to expand and see the Window Constructor

In the Window class is also a function that makes it easier to set a Scene to my window.

```
    public void setScene(Scene scene){
        this.scene = scene;
        scene.load();
        this.revalidate();

        this.addKeyListener(new KeyAdapter() {
            @Override
            public void keyPressed(KeyEvent e) {
                if(e.getKeyCode() == KeyEvent.VK_ESCAPE) {
                    System.exit(0);
                }
            }
        });
    }
```

## Scene class

In the Scene class the constructor has two arguments. The first is the window, on wich window the scene should be loaded. The second is the backgorund. There are also some functions like:

- add(), makes it possible to add components
- load()
- loadBackground()

  ```
  public void loadBackground(URL imgPath){

          Image originalImg = new ImageIcon(imgPath).getImage();
          Debugging.log(String.format("Loading Background: %s",Assets.absolute(Assets.Backgrounds[_stage])));
          ((JFrame) window).setContentPane(new ImagePanel(originalImg, originalImg.getWidth(null), originalImg.getHeight(null)))

          try {
              BufferedImage bim = Assets.getBufferedImage(Assets.Backgrounds[_stage]); // Color
              int sample        = bim.getRGB(0, bim.getHeight() - 1);
              Color floor       = new Color(sample, true);
              floorColor        = floor;
              this.window.getRootPane().setBackground(floor);
          } catch (Exception e) {
              Debugging.error(String.format("Can't change floor color! (%s)", e.getMessage()));
          }

      }
  ```

  The loadBackground() function firsts gets the imagePath. Writes a message in the console. Creates a new element and adds the Picture. After that it changes the backgroundcolor to the lowest left pixel of the loaded image.

## Level

First we set the window and the stage of a scene. Than we create a Puzzle. After that we add a simple Diaolog and add a option to continue or else. Than we display the puzzle and destroy the dialog. If you lose and you did not solve the puzzle, a deathscene gets displayed. Otherwise you can go on to another level. I also add an Enemy with its position and call the think() function if you click on the Player. Here is an example for a level:

▶ Click to expand

# NPC and Enemy

## NPC

We have following functions in NPC: * NPC(int, Scene, boolean, Rectangle, Runnable), constructor * destroy(), destroys the NPC * flip(boolean), flips the NPC to the right or left * paintComponent(Graphics), gets called(from JAVA) if something changes on the screen. The function repaints the NPC after an event.
* resize(), resizes the NPC and Speech if the window gets resized (if it is not fullscreen) * say(String), gets called by a "Thought" * setCallback(Runnable), handels input when a button is clicked

▶ Click to expand the NPC class

## Enemy The Enemy class is very simmilar to the NPC class but an Enemy can not think like the NPC. Both entities are drawn behind a transparent button. The

Enemy class has following functions:

- Enemy(int, Scene, boolean, Rectangle, Runnable), constructor
- destroy(), destroys the Enemy after an Event and gets repainted by the paintComponent(Graphics) function
- paintComponent(Graphics), paints the component. In this case the entity
- resize(), resizes the enemy after the window is resized

- setCallback(Runnable), handles input after the button on the Enemy is clicked

  ▶ Click to expand the Enemy class

## Enemy

```
 public class Enemy extends JButton
{

    private int eid;
    private Scene scene;
    private Enemy _this;
    private boolean flipImg;

    private int originalImageHeight;
    private int originalImageWidth;
    private Rectangle originalRect;

    private Image image;
    private Runnable callback; //pressed ene

    @Override
    protected void paintComponent(Graphics g) {
        super.paintComponent(g);
        int windowHeight  = (int) super.getHeight();
        int windowWidth   = (int) super.getWidth();
        int imageHeight   = (int) originalImageHeight;
        int imageWidth    = (int) originalImageWidth;
        double multiplier = (double) windowHeight / imageHeight;

        int imageOverflow = (int) Math.floor((imageWidth * multiplier) - windowWidth);
        if (windowHeight <= 0 || imageHeight <= 0){

            Debugging.warn("Invalid Sized for Background Scaling!");
            return;

        } else {

            if (flipImg){
                g.drawImage(image, -((int) imageOverflow / 2) + (int) Math.round(imageWidth * multiplier), 0, -((int) Math.round(ima
            } else {
                g.drawImage(image, -((int) imageOverflow / 2), 0, (int) Math.round(imageWidth * multiplier), (int) Math.round(imageH
            }

        }
    }

    private void resize(){

        double xMultiplier     = scene.window.getWidth() / Assets.refrenceRect.getWidth();
        double yMultiplier     = (scene.window.getRootPane().getHeight() - Assets.Scalings.get("dialog-section")) / Assets.refrenceR

        int newX               = (int) (originalRect.getX() * xMultiplier);
        int newY               = (int) (originalRect.getY() * yMultiplier);

        int newW               = (int) (originalRect.getWidth()  * yMultiplier);
        int newH               = (int) (originalRect.getHeight() * yMultiplier);

        Rectangle relativeRect = new Rectangle(newX, newY, newW, newH);
        setBounds(relativeRect);
    }

    public void setCallback(Runnable callback){

        this.callback = callback;

    }

    public void destroy(){

        this.getParent().remove(this);
        scene.window.repaint();

    }

    public Enemy(int eid, Scene scene, boolean flip, Rectangle rect, Runnable callback)
    {

        Image img                 = new ImageIcon(Assets.absolute(Assets.Enemies[eid])).getImage();
        this.originalImageHeight = img.getHeight(null);
        this.originalImageWidth  = img.getWidth(null);
        this.image               = img;
        _this                    = this;
        this.eid                 = eid;
        this.scene               = scene;
        this.flipImg             = flip;
        this.originalRect        = rect;
        this.callback            = callback;

        Debugging.log(String.format("Loading game.Enemy Sprite: %s (ID: %s)",Assets.absolute(Assets.Enemies[eid]), this.eid));

        final Scene _scene = scene;
        addMouseListener(new java.awt.event.MouseAdapter() {
            @Override
            public void mousePressed(MouseEvent arg0) {

                if (Assets.level_editor && arg0.getButton() == 1){
                    _scene.window.le.startDragging(_this, arg0.getX(), arg0.getY());
                    _scene.window.dispatchEvent(arg0);
                } else {
                    if (!_scene.interactionsDisabled){

                        if (_this.callback != null){
                            _this.callback.run();
                        }

                    }
                }
                super.mousePressed(arg0);
            }

        });

        setOpaque(false);
        setFocusable(false);
        if (!Assets.hitboxes){

            setBorder(null);

        }
        setContentAreaFilled(false);

        scene.window.addComponentListener(new ComponentAdapter() {
            public void componentResized(ComponentEvent componentEvent) {
                resize();
            }
        });
        resize();

        scene.add(this);

    }

}
```

# Puzzle

## ImagePuzzle

- ImagePuzzle(URL, Scene, int, int), constructor. Arguments are [ImageURL], mostly the actual scene, width of the puzzle, height of the puzzle
- check(), checks if the puzzle is solved
- destroy(), destroys the puzzle
- disable(), "unclicks" all buttons/pieces at the beginnig
- resize(), resizes the puzzle after the window gets resized
- revisualize(), "repaints" the puzzle (e.g. after a piece is swapped)
- select(), selects a piece
- shuffle(), shuffle the puzzle after the inspcetion phase
- start(int, Runnable, Runnable), starts the puzzle and creates a time bar below the puzzle

- switchPieces(), switches two pieces next to each other (not over cross) after they got clicked

  When a puzzle gets created, we give the constructor the number of how many divisions we want.

  ````JAVA public ImagePuzzle(URL image, Scene scene, int size, int divisions){ ... }
  ````

~~~
 . Than we create a new Gridlayout out of buttons.
If a button gets clicked, the boarder changes color to yellow (unclicked is it grey). After the second button is clicked we


This is the start() function form the class ImagePuzzle. It gets called to start the puzzle.
````JAVA
public void start(int time, Runnable winCallback, Runnable loseCallback){

    if (started == true){

        Debugging.warn("Tried to start puzzle that already had been started prior.");
        return;

    }
    started = true;

    setVisible(true);
    progress.setVisible(true);

    Timer t1 = new Timer();
    t1.schedule(new TimerTask() {
        @Override
        public void run() {

            SwingUtilities.invokeLater(new Runnable() {
                public void run() {
                    shuffle();
                }
            });
            t1.cancel();
            Timer t = new Timer();
            timer   = t;
            long finishTime = System.currentTimeMillis() + time;
            t.schedule(new TimerTask() {
                @Override
                public void run() {
                    long currentTime = System.currentTimeMillis();
                    if (finishTime - currentTime <= 0L){
                        progress.setBounds((int) progress.getBounds().getX(), (int) (getBounds().getY() + getBounds().getHeigh

                        if (loseCallback != null){

                            loseCallback.run();
                            disable();

                        }

                        t.cancel();

                    } else {

                        long timeLeft = finishTime - currentTime;
                        percent       = 100d / time * (double) timeLeft;
                        int prog      = (int) Math.floor((double) getBounds().getWidth() / time * timeLeft);
                        progress.setBounds((int) progress.getBounds().getX(), (int) (getBounds().getY() + getBounds().getHeigh

                    }
                }
            }, 0, 10);

        }
    }, prepTime);
    this.win = winCallback;

}
~~~

PuzzlePiece

- PuzzlePiece(Image, int, int, int, ImagePuzzle), constructor
- paintComponent(Graphics),
- select(), makes the border yellow after recieving the information about the clicked piece from ImagePuzzle.select() function
- unselect(), makes the border normal (grey) after recieving the information about the clicked piece from ImagePuzzle.select() function.

▶ Click to expand class diagram

Goals

The goals of the Project were to improve my JAVA (I ussualy program in C++) and make a playable game. I had alot of fun and the game could easily get expanded by adding fighting enemies, making the player moveable, adding a storry and adding Items. I also wanted to improve and understand how simple games are working and how to design a game that contains readable sourcecode.

Problems

There were a few problems like bugs and a lot of error messages. One main problem was the time managment. I wasted at the beginning of the project alot of time by doing nothing.

Source

- https://stackoverflow.com/
- https://www.geeksforgeeks.org/java/?ref=shm
- https://deadcells.fandom.com/wiki/Dead_Cells_Wiki
- https://deadcells.fandom.com/wiki/NPCs
- https://deadcells.fandom.com/wiki/Biomes

- https://docs.oracle.com/javase/tutorial/uiswing/index.html

  There were defintily a lot more sources but I didnt write them down and the soruces above are the sources I mainly used. And for the motivation I heard a lot of music form chilled-cow and I drank a lot of coffee and black tea.

  A friend of my old class supported me by answereing me some questions about JAVA-swing : Marc