HeroQuest - Step by Step

DISCLAIMER!

This is a collection of notes I had made while programming the game, so it is rather confusing.

A) MAIN and MENU

1. Writing **Game.java** (Main) --> creates a **frame** by referring to settings defined in GamePanel
2. Writing **GamePanel.java** (Main) --> I implement a **JPanel** (set width, height, scale), I implement THREAD and KEYLISTENER, for init() update() and draw() (and keyevents) I refer to GameStateManager (gsm).
3. **GameStateManager.java** (GameState)--> is used to manage the different states, **setState** to change between states (uses loadState and unloadState), constructor initializes the current state to MENUSTATE, update(), draw(), keyevent
4. **GameState.java** (GameState) --> create an **abstract** class with methods to be used by subclasses (init, update, draw, keypressed, keyreleased)
5. **Background.java** (TileMap) --> in the constructor we insert the way to load BufferedImage from an InputStream (package javax.imageio.ImageIo) , scrollable background
6. **MenuState.java** (GameState) --> initialize options, create a Background object in the **constructor** that scrolls, **draw()** for title and options (with evidence of current choice), **select()** to implement choice, **keyPressed()** to move in the menu
   (extends GameState)

B) TILEMAP

1. **Tile.java** (TileMap) --> used to define the image and type of the individual Tile (which can be either NORMAL or BLOCKED)
2. **TileMap.java** (TileMap) --> after defining the variables (knowing that we don't want to load the whole map right away, but one piece at a time), we write the constructor that takes into account the number of rows and columns to display according to the size of the screen. loadTiles**()** to load the individual tiles, and **loadMap()** to load the .map file, **draw()** to actually draw the map according to **SetPosition()** that follows the player "slowly" and calculates colOffset and rowOffset.
3. **Level1State.java** (GameState) --> extends GameState, constructor calling **init()**, in which I create the **TileMap and a Background**, which I draw in draw, in update() only the background bg.

C) MAP OBJECT SUPERCLASS

1. **MapObject.java** (Entity) --> abstract, lots of variables, **intersects(MapObject)** which calculates collisions using getRectangle(), **checkTileMapCollision()** to check if the

character as it moves will encounter an object (or if it will fall), it makes use of the calculateCorners function that checks if they are BLOCKED (and therefore there may be a collision).

setPosition(x,y) for the global position, setMapPosition() for where to draw the object, notOnScreen() to check if the object is on the screen,
draw() to draw it (facingRight normal, else flipping the sprite), LOCAL POSITION: (x + xmap, y + ymap)

D) PLAYER

1. **Animation.java** (Entity) --> setFrames**(BufferedImage[] frames)** starts the animation from 0, **update()** checks the time elapsed between frames and possibly starts the next frame, then checks if the animation is finished (boolean playedOnce).
2. **Player.java** (Entity) --> I create a list with animations (array of frames) and associate them with actions (numbered).
3. **Player Builder** --> I set the values of the variables and create a list for the FireBalls, since there can be more than one on the screen. I read the InputSteam of the sprites and create a LIST of animations (list of BufferedImage[]). For each of the 7 animations, I load the respective frames (taken from the spritesheet through getSubimage()) into the 7 arrays and load them one by one into the list. **And I set the starting animation (IDLE).**
4. **update()** --> various checks (on position and collisions) and a series of IFs to SET ANIMATIONS and finally call **animation.update()**
5. **draw()** --> FLINCHING; super.draw(g) from the superclass MapObject
6. **getNextPosition()** --> called right at the beginning in the constructor. This function defines **movement** (whether it goes to the right, or left, or is stationary), defines **JUMPING** and **FALLING**, and the fact that it cannot move while attacking.
7. **Level1State.java** --> I create the Player in init() and set the position, update() and SetPosition to make the camera follow the Player. And draw() with the functions called by the Player.

Finally I set the KeyEvents for keyPressed and keyReleased (setLeft, setRight, setUp, setDown, setJumping, setGliding etc...)

E) PLAYER ATTACKS

1. **Fireball.java** (Entity) --> in the constructor I set the values of the variables and load the sprites (for motion and hit animations). setHit and update().
2. **Player.java** (Entity) --> I define the arraylist of fireballs which I initialize in the constructor. In update() I insert fireball attack, where I regenerate energy continuously, and create the new fireball (adding it to the list) checking if I have enough energy. I update draw().
   Also in update() I update the fireball (calling it from Fireball.java) and check if I have to remove it from the list (and thus from the game).

F) FIRST ENEMY

1. **Enemy.java** (Entity) --> Enemy can do contact damage to the player, hit(int damage) to be used when the enemy receives damage.
2. **Slugger.java** (Entity.Enemies) --> set variable values, load sprites and create motion animation. update() controls motion (via getNextPosition()), flinching control, if it hits a wall it comes back, update animation. draw(), but only if it's on the screen.
3. **Level1State.java -->** I define an arraylist of enemies and insert them with the populateEnemies() method, update draw() and update(), use loops to check all enemies still in the list.
4. **HUD.java** (Entity) --> just the constructor where I load the images and draw(Graphics2D g) with font setting etc.
5. **Level1State.java** --> create hud and update draw()

G) ATTACKING ENEMIES

1. **Player.java --> checkAttack(ArrayList<Enemy> enemies)** I check the various interactions, if scratching (right or left) has hit an enemy and then I inflict damage (e.hit()), same for fireballs, and I check if an enemy has hit the Player (intersects) and I inflict damage to the Player. I define the hit function for the player.
2. **Explosion.java** (Entity) --> Enemy death animation
3. **Level1State.java** --> I insert Explosions as ArrayList, in update all enemies when an enemy dies I add a new explosion to the list at the point of death, add in update explosions and update draw().

H) MUSIC AND SOUND EFFECTS

1. **I use 3 libraries:** jl1.0.1.jar / mps3spi1.9.5.jar / tritonus_shar.jar
2. Add Jars from configure Build Path in Project
3. SFX --> sound effects, Music --> level music
4. **AudioPlayer.java** (Audio) --> load the audio file, decode it and take the clips, play and stop methods
5. **Level1State.java -->** play the music of the level, for SFX use HashMap

---

1. Added HelpState
2. Added PauseState

3Added java keys in package Handlers