True Hero

By Taga Dan-Claudiu, 1208B

Gameplay:

Single player game where the player must defeat or avoid monsters and collect potions in order to get stronger and defeat the Final Boss.

Gameplay involves exploring two levels packed with monsters to avoid or fight and potions to collect and drink. The player gains score upon defeating enemies and picking up potions, but only gains experience points upon slaying monsters of varying strengths.

The player has to manage two stats: his health points and his mana points, which he can replenish with potions. He also has to carefully pick his battles: stronger foes protect valuable potions and yield useful exp points, but are dangerous beasts.

The player can also level up once he gets enough exp points, increasing his stats and becoming stronger. At the end of his journey lies the true test: the final boss. His defeat is the only way to win the game.

Plot:

The game is set inside the tomb of an ancient evil being, slain by a legendary hero of the past. The monster has been dead for thousands of years, only to be mysteriously revived. Weakened from decomposing for so long, he summons his minions to defend his mortal body and corrupts the guardians of his grave to serve a new master: the evil they swore to keep locked in this tomb.

After a few years, the Legendary Evil has almost grown his past strength, until a mysterious new being arrives in the wretched place. Weaponless, covered from head to toe with dark fabric; only his piercing red eyes are seen: an unnatural color that only speaks danger.

Corrupted guards stand before him to stop him, but in the blink of an eye, they lie dead on the ground, with burnt fur and evaporated blood: a mage!

The dark minions protecting the resting place of their master take notice, and begin patrol to find and kill this intruder. Upon entering the tomb, the nameless mage finds various potions on the ground: old artifacts left behind the legendary hero of the past, now corrupted by the magic of the awakened beast.

He picks up a potion, which's content can easily turn any man into a puppet, and chugs down the contents, only to feel their positive outcome. What exactly is this nameless mage...?

His arrival has been noticed by the Legendary Evil, as it fortified his ranks of minions to stop the incoming danger and to keep the cursed potions made by his past enemy from the hands of his invader.

With determination and hunger in his eyes, the nameless mage slays minion after minion, getting stronger with each kill. Slowly, but surely, his prey is in hand's reach: two more rooms left.

Surely, the death of a revived evil being is a good thing, and the slayer should be a hero. But what are the intentions of this nameless mage? What does he plan to do with the newfound power from slaying this ancient evil? Who exactly is the True Hero here...?

The eyes of the mysterious mage glow blood red, as he looks towards his objective. Underneath his mask, the lines of a smirk can be seen.

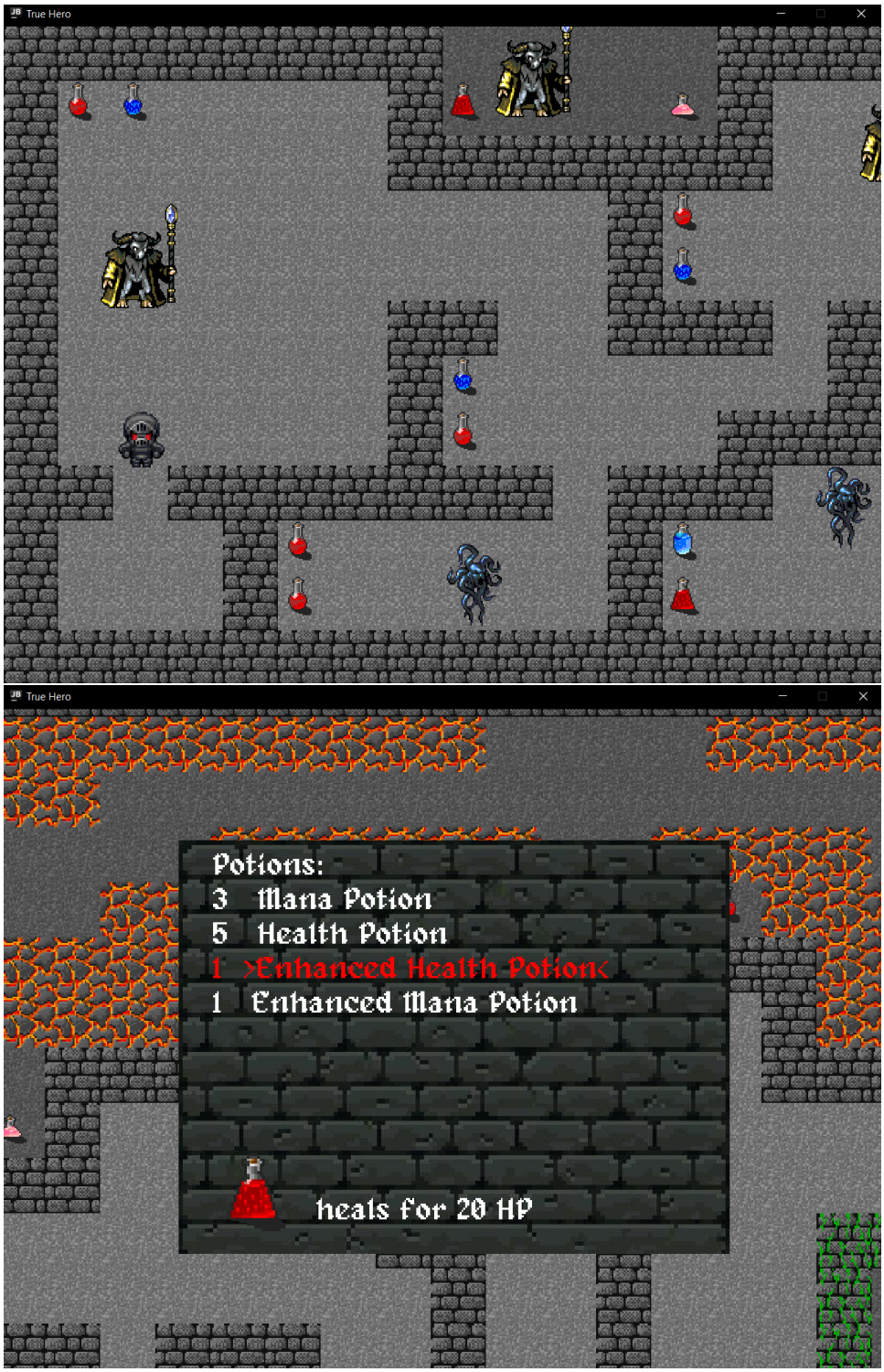
Characters:

I. **The Nameless Mage** is the mysterious protagonist and player-character. Not much can be known about him, as he is covered completely in dark-colored armor and fabric. Only visible features are his intense red eyes, a feature of no normal man, especially not one of a hero, but a beast's. He has the ability to control arcane magic and consume essence of his enemies (exp), making himself stronger.
II. **Horned Magus** is part of an old defender bloodline of the world from evil. They were originally protecting the tomb, but failed their mission, as they all got corrupted or killed by the revival of the Ancient Evil. Now puppets, they lost a whole lot of their former power and serve their new master mindlessly.
III. **Legendary Evil's Servants** are fragments of Legendary Evil's will manifested into reality. They slay or corrupt living beings that don't align with their master's wishes.
IV. **Demon Born of Shadows** is a dangerous monster created from the corpses of the old Horned Magi by the Legendary Evil upon The Nameless Mage's arrival. Ruthless but fragile, they roam the halls of the tomb looking for the intruder, and with their sharp claws they try to rush him down with a series of powerful cuts.
V. **Protector of the Tomb** is a tentacle-like monster created to restrict certain paths and keep any intruder at bay with their powerful body. Resistant body and thick skin, these creatures can take powerful blows. Their attacks shouldn't be underestimated either, as the fight with this creature is an endurance test.

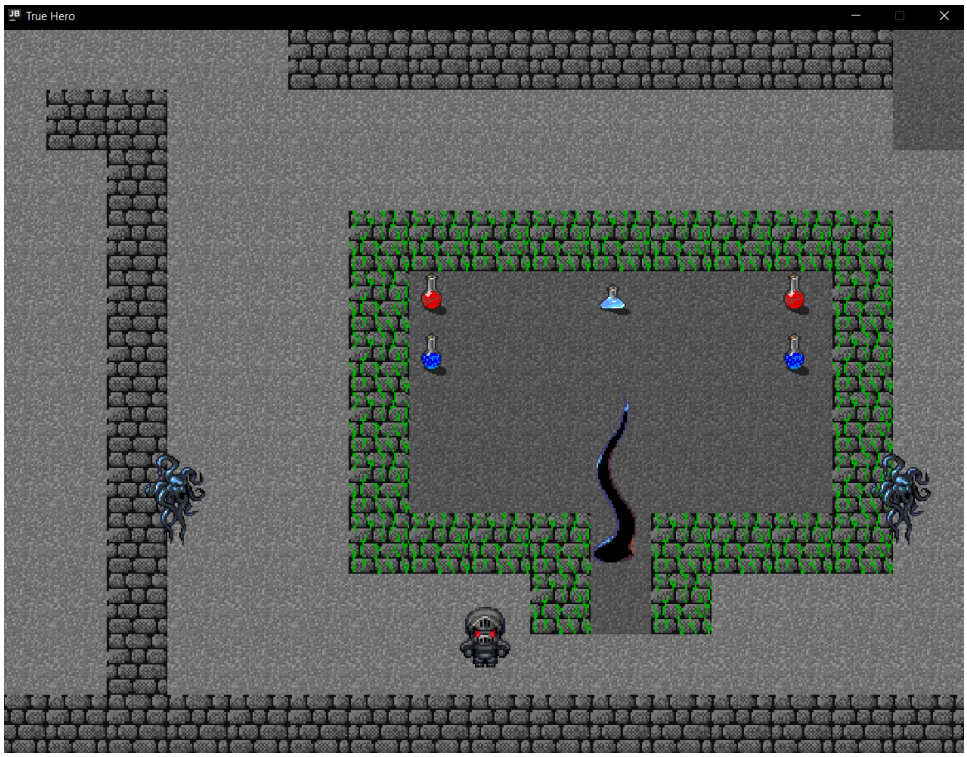
VI. **Legendary Evil** is the ancient being slain thousands of years ago by an incredibly powerful human hero, left to rot inside this tomb defended by Ancient Magi. Now reborn, its will manifested into corrupting and conquering the tomb, the place where it would rest and regain its former power. As the process of healing is almost done, it has almost regained its old forms and names.

Mechanics:

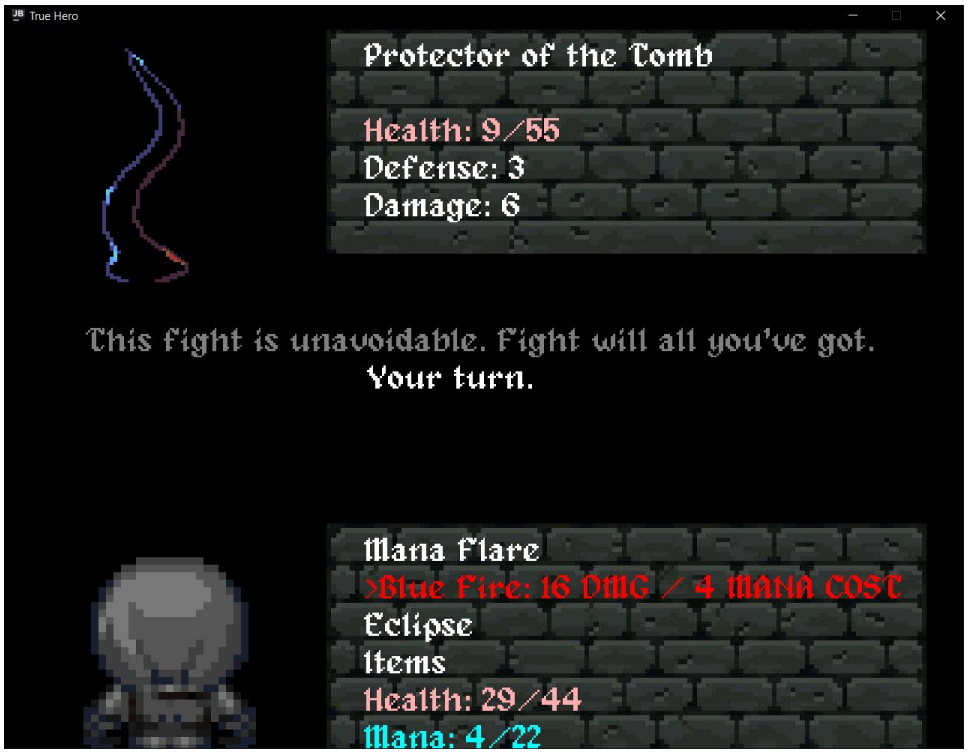
There are two states of the game. Exploration and Combat. During Exploration, the player can move around, pick up potions, open up their inventory and stats page where they can see their stats and drink their potions.



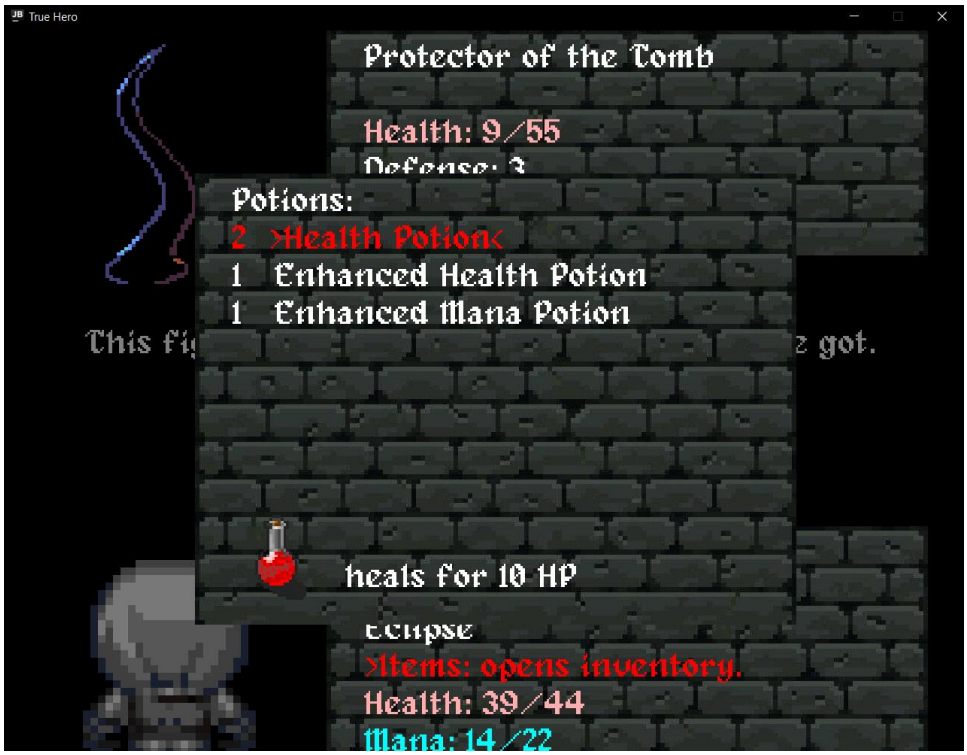
The enemies patrol in different patterns, and can mostly be avoided. It is recommended to fight them though, in order to get stronger and to have access to helpful potions.



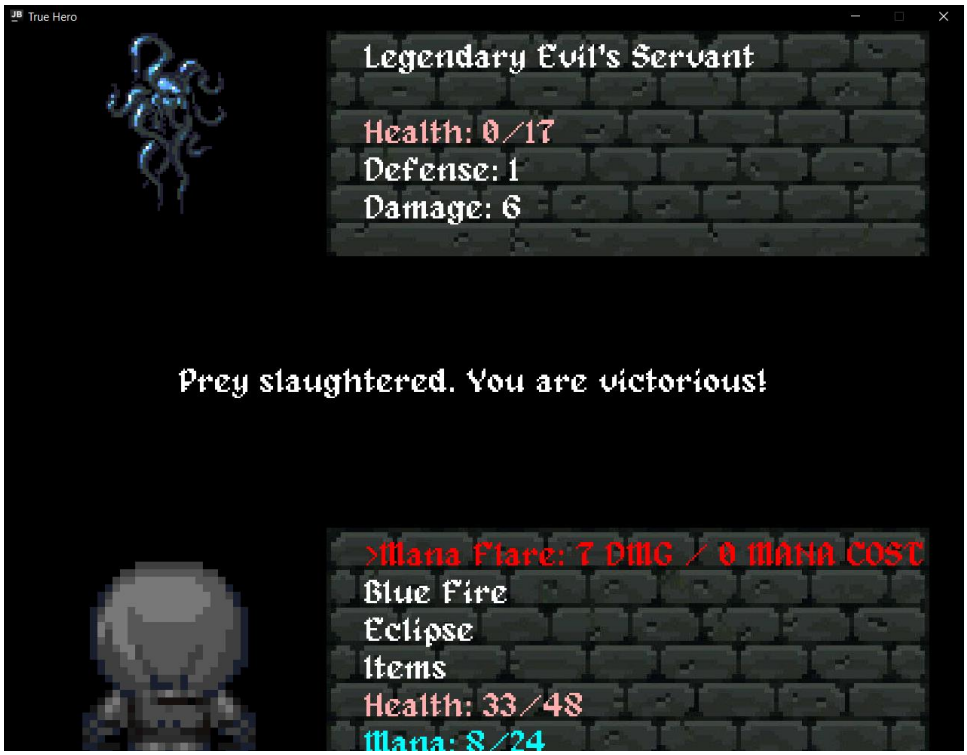
Once the player gets too close to an enemy, they enter the Combat together, where the player gets introduced into the turn-based combat system. During the player's turn, they get to choose between 4 options: 3 different forms of attack and access to inventory. The first form of attack is the weakest but costs 0 mana, while the third one deals the most damage but costs the most mana (at most 10 mana per spell). During their turn, the player gets information about their stats and the monster's stats. Every monster has 3 stats: Health, Attack and Defense. Health and Defense influence the durability of the enemy in battle, while the attack displays their offensive power.



The player has these 3 stats as well, but with 2 more: Mana and Intelligence. Mana is an important resource that allows the use of more powerful spells, and Intelligence is a stat that will grow the potency of the player's attack spells, even reducing cost. It's very important to manage health and mana properly, as they can only be replenished by (limited number of) potions or level up. The player can access their inventory during their turn, to consume potions in a pinch, but this costs their turn.



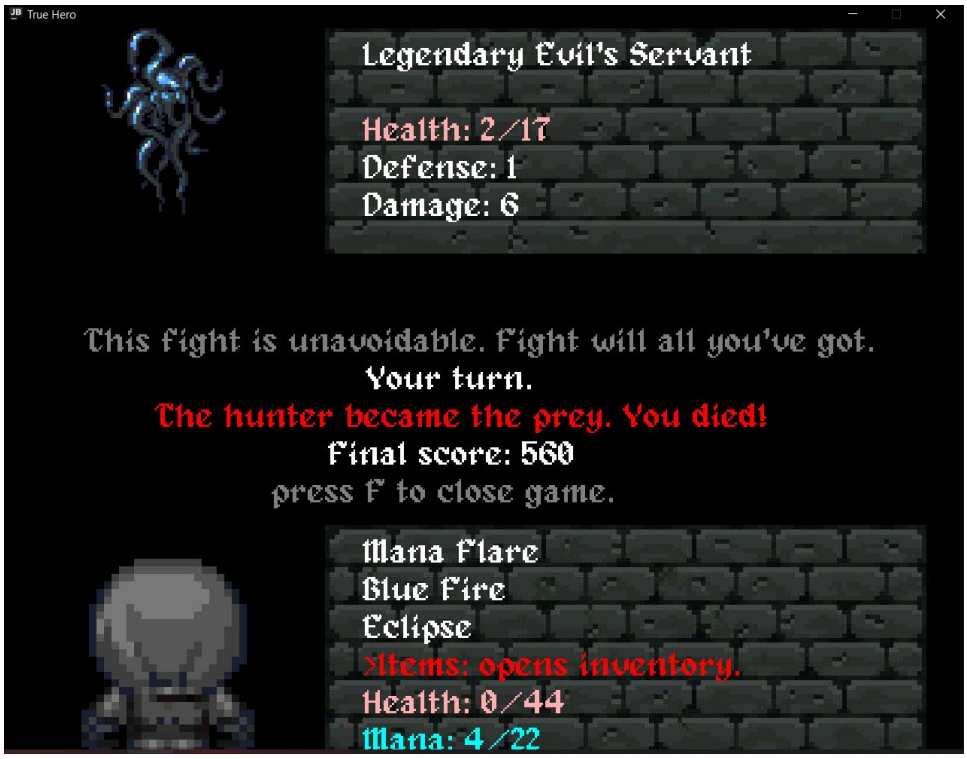
After the player completed his action, it's the enemy's turn to act. They will attack the player, dealing damage equal to their Attack stat, subtracted by the player's Defense stat, then ends their turn. This will repeat until one of the two dies. If the player's health reaches 0, they die and the game will be over, showing their total score and asking them to press f to close the game. If the monster's health reaches 0, they die and if the enemy is not the final boss, the player wins the encounter, returning to Exploration and being awarded EXP and Score Points.



The player starts with level 1, and can gain EXP to increase the level, increasing their stats, the most important stats that get increased being Intelligence and Defense. The player also gains Score Points upon defeating enemies and collecting potions.

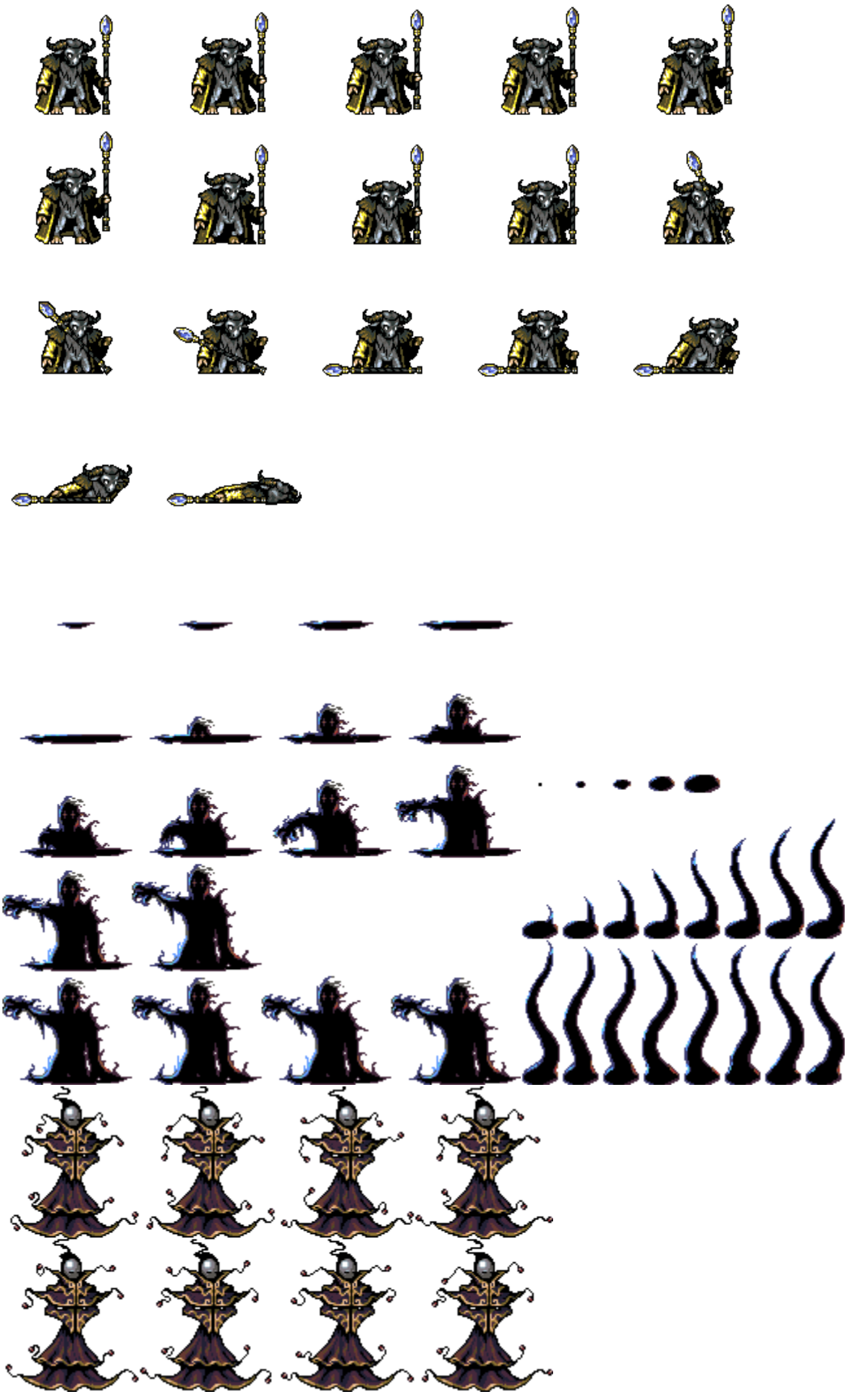
The player can only win the game by defeating the Legendary Evil, the most powerful enemy of the game.

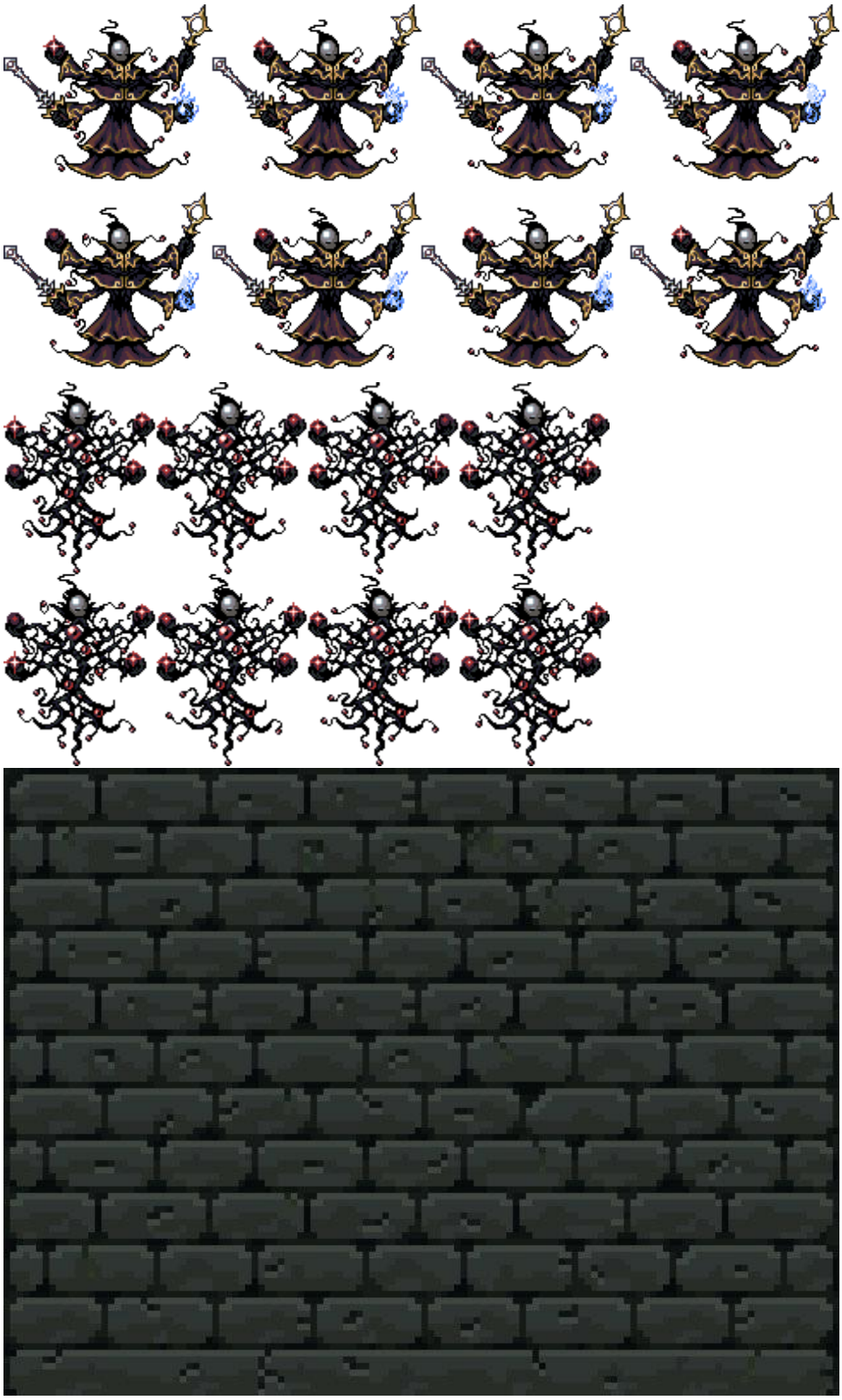
Defeating the Legendary Evil grants the player a remarkable amount of score points. During combat, the player *CANNOT* escape the encounter: it's a battle to the death.



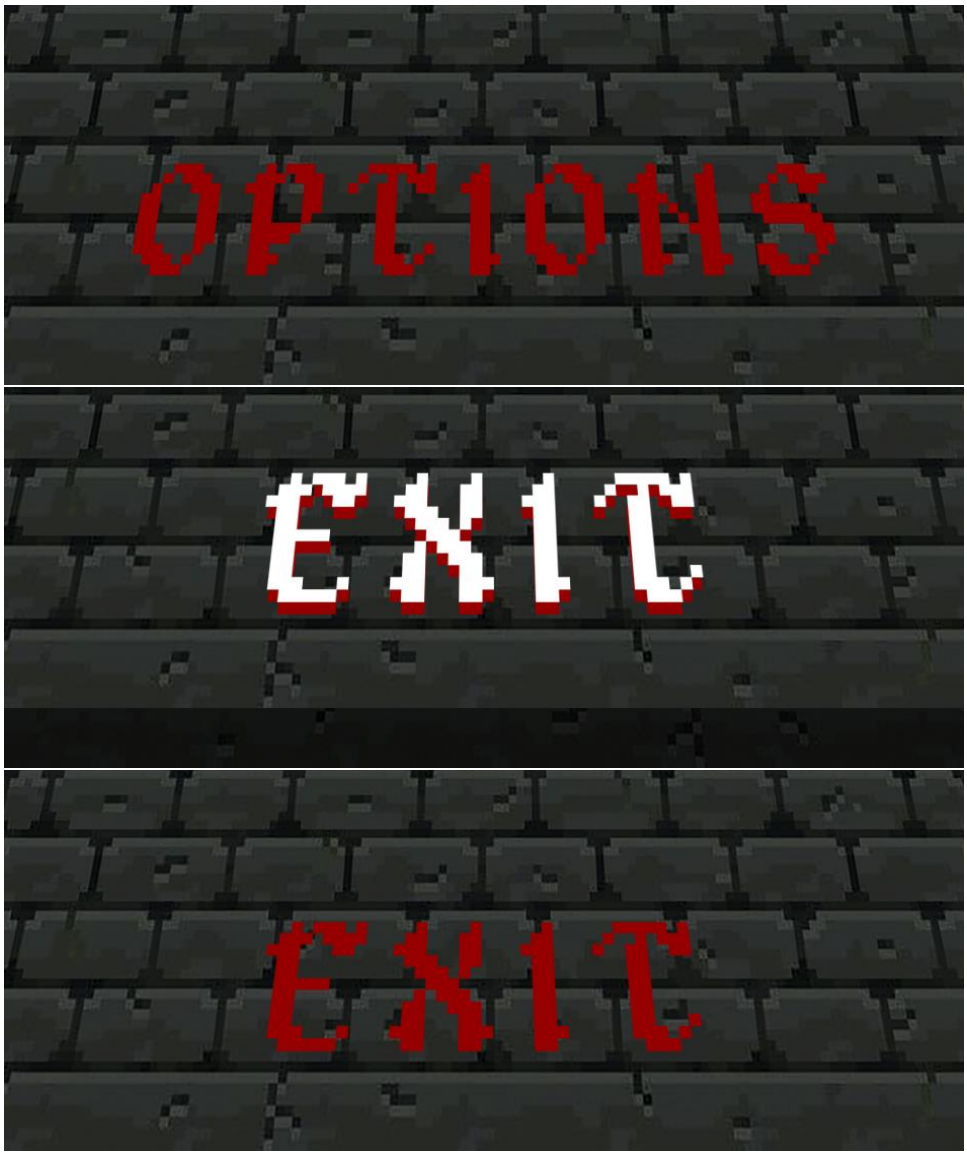
Sprite Sheets:


Assorted
RE Sprites
by:
ShadowLeggy
v1.0
doubleleggy.deviantart






START
START
OPTIONS

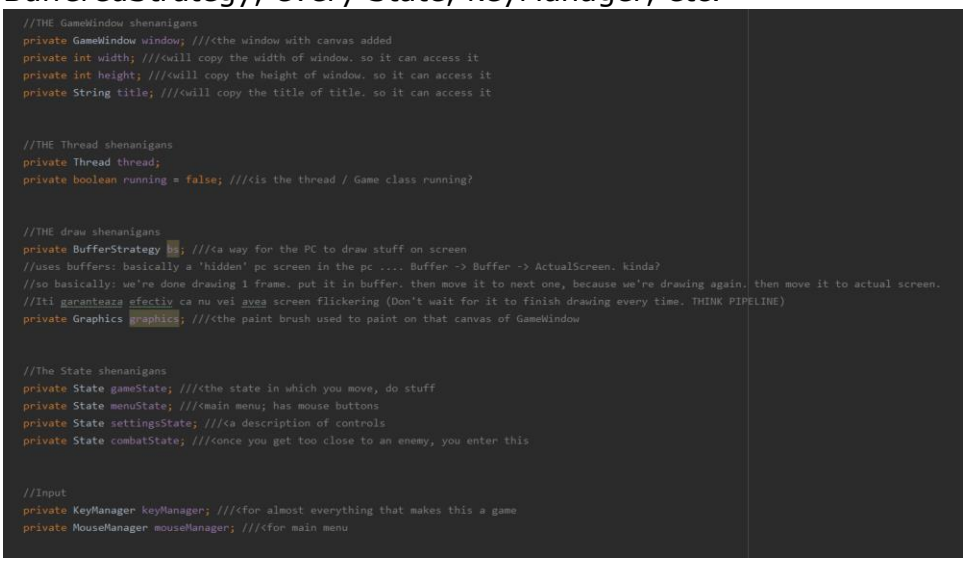


Technical Documentation:

The project has many important classes that assure proper functioning. The main classes of the project are: Main, Game, World, Assets, Entity, Tile, GameWindow, KeyManager, State, Handler, Player.

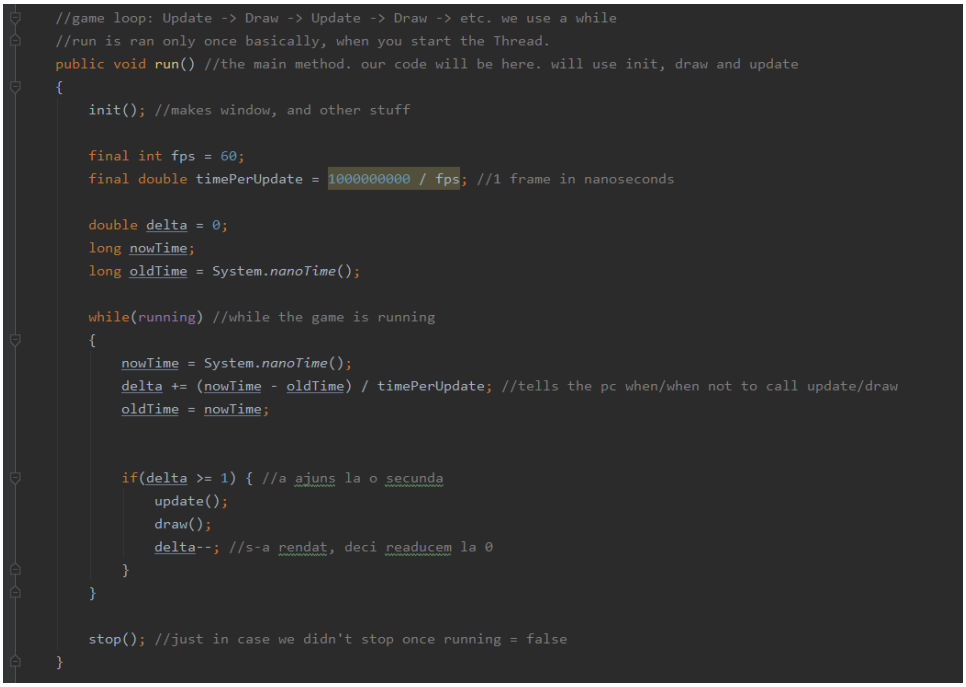
The Game class is the main class of the entire project, containing every necessary member for the game to run, including Graphics, GameWindow, Thread,

BufferedStrategy, every State, keyManager, etc.



```
//THE GameWindow shenanigans
private GameWindow window; ///<the window with canvas added
private int width; ///<will copy the width of window. so it can access it
private int height; ///<will copy the height of window. so it can access it
private String title; ///<will copy the title of title. so it can access it


//THE Thread shenanigans
private Thread thread;
private boolean running = false; ///<is the thread / Game class running?


//THE draw shenanigans
private BufferStrategy bs; ///<a way for the PC to draw stuff on screen
//uses buffers: basically a 'hidden' pc screen in the pc .... Buffer -> Buffer -> ActualScreen. kinda?
//so basically: we're done drawing 1 frame. put it in buffer. then move it to next one, because we're drawing again. then move it to actual screen.
//Iti garanteaza efectiv ca nu vei avea screen flickering (Don't wait for it to finish drawing every time. THINK PIPELINE)
private Graphics graphics; ///<the paint brush used to paint on that canvas of GameWindow


//The State shenanigans
private State gameState; ///<the state in which you move, do stuff
private State menuState; ///<main menu; has mouse buttons
private State settingsState; ///<a description of controls
private State combatState; ///<once you get too close to an enemy, you enter this


//Input
private KeyManager keyManager; ///<for almost everything that makes this a game
private MouseManager mouseManager; ///<for main menu
```

This class initializes the project and is the main thread, having the only run() method in the entire architecture. It also contains the Update() -> Draw() -> Update() -> Draw() -> … loop, and also assures a framerate of 60 fps.



```
//game loop: Update -> Draw -> Update -> Draw -> etc. we use a while
//run is ran only once basically, when you start the Thread.
public void run() //the main method. our code will be here. will use init, draw and update
{
    init(); //makes window, and other stuff

    final int fps = 60;
    final double timePerUpdate = 1000000000 / fps; //1 frame in nanoseconds

    double delta = 0;
    long nowTime;
    long oldTime = System.nanoTime();

    while(running) //while the game is running
    {
        nowTime = System.nanoTime();
        delta += (nowTime - oldTime) / timePerUpdate; //tells the pc when/when not to call update/draw
        oldTime = nowTime;


        if(delta >= 1) { //a ajuns la o secunda
            update();
            draw();
            delta--; //s-a rendat, deci readucem la 0
        }
    }

    stop(); //just in case we didn't stop once running = false
}
```

Main class is pretty simple, with only 1 method: main(), having a local Game variable that is run with the start() method.

The GameWindow class is the skeleton of the game, containing two very important members: JFrame and Canvas. With them the entire game will be drawn, thus shown to the player.

```
private void createWindow()
{
    if (frame != null)
        return;

    frame = new JFrame(title);
    frame.setSize(width, height);
    frame.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE); //so it won't sit in the BG

    frame.setResizable(false); //can't drag the window
    frame.setLocationRelativeTo(null); //window appears in the center of screen
    frame.setVisible(true); //frames aren't visible by default

    canvas = new Canvas();
    canvas.setPreferredSize(new Dimension(width, height));
    canvas.setMaximumSize(new Dimension(width, height));
    canvas.setMinimumSize(new Dimension(width, height));
    //makes sure you keep the canvas within those dimensions

    canvas.setFocusable(false);
    //super important. so only the JFrame has focus.

    frame.add(canvas); //add the canvas to frame
    frame.pack(); //resize window to see canvas fully
}
```

The abstract class State with its 4 extended classes separates the game into different states: menu, game, combat, settings. Only one state can be active at a time, while the other three "wait their turn" until it's possible to change the current state.

Menu state is the main menu of the game, the first thing the player sees when they open the game. It gives the possibility to the player to choose between starting the game, opening settings or exiting game.

Game state is the first half of the game, in which the map and entities, including the player, appear. The player may move, open their inventory, pick up objects, fight other moving entities.

Combat state is the second half of the game, and it appears when the player character gets too close to an enemy. The combat is turn-based, the two entities taking turns on hitting on another, until one dies, which results to either the game continuing or ending (game over).

Settings state is a part of the menu in which the player is shown the control keys.

The Assets class contains the entire graphical part of the game, using sprite sheets to load images, animations or fonts. These images are stored in BufferedImage / BufferedImage[] members, which will be used by entitites or tiles, to give life to the game.

```
factory = new SpriteSheetFactory();
SpriteSheet sheetPlayer = factory.makeSheet(ImageLoadFunction.loadImage( path: "/textures/player.png"));
SpriteSheet sheetGrounds = factory.makeSheet(ImageLoadFunction.loadImage( path: "/textures/dg_grounds32.gif"));
SpriteSheet sheetWalls = factory.makeSheet(ImageLoadFunction.loadImage( path: "/textures/dg_dungeon32.gif"));
SpriteSheet sheetPotions = factory.makeSheet(ImageLoadFunction.loadImage( path: "/textures/potions_dungeon.png"));
SpriteSheet sheetBlueFlame = factory.makeSheet(ImageLoadFunction.loadImage( path: "/textures/Monsters/flameball-32x32.png"));
SpriteSheet sheetTitle = factory.makeSheet(ImageLoadFunction.loadImage( path: "/textures/UI/titlewhite.png"));

//Monsters
SpriteSheet sheetHornedMonster = factory.makeSheet(ImageLoadFunction.loadImage( path: "/textures/Monsters/gnu-120x100.png"));
SpriteSheet sheetShadowMonster = factory.makeSheet(ImageLoadFunction.loadImage( path: "/textures/Monsters/shadow-80x70.png"));
SpriteSheet sheetMinionMonster = factory.makeSheet(ImageLoadFunction.loadImage( path: "/textures/Monsters/minion-45x66.png"));
SpriteSheet sheetTentacleMonster = factory.makeSheet(ImageLoadFunction.loadImage( path: "/textures/Monsters/tentacles-25x90.png"));
SpriteSheet sheetBoss1 = factory.makeSheet(ImageLoadFunction.loadImage( path: "/textures/Monsters/mage-1-85x94.png"));
SpriteSheet sheetBoss2 = factory.makeSheet(ImageLoadFunction.loadImage( path: "/textures/Monsters/mage-2-122x110.png"));
SpriteSheet sheetBoss3 = factory.makeSheet(ImageLoadFunction.loadImage( path: "/textures/Monsters/mage-3-87x110.png"));
```

```
//Player:
player_down = new BufferedImage[2];
player_up = new BufferedImage[2];
player_left = new BufferedImage[2];
player_right = new BufferedImage[2];

playeridle_down = sheetPlayer.crop( x: tileWidth*10,  y: 0, tileWidth, tileHeight);
player_down[0] = sheetPlayer.crop( x: tileWidth*9,  y: 0, tileWidth, tileHeight);
player_down[1] = sheetPlayer.crop( x: tileWidth*11,  y: 0, tileWidth, tileHeight);

playeridle_left = sheetPlayer.crop( x: tileWidth*10, tileHeight, tileWidth, tileHeight);
player_left[0] = sheetPlayer.crop( x: tileWidth*9, tileHeight, tileWidth, tileHeight);
player_left[1] = sheetPlayer.crop( x: tileWidth*11, tileHeight, tileWidth, tileHeight);

playeridle_right = sheetPlayer.crop( x: tileWidth*10,  y: tileHeight*2, tileWidth, tileHeight);
player_right[0] = sheetPlayer.crop( x: tileWidth*9,  y: tileHeight*2, tileWidth, tileHeight);
player_right[1] = sheetPlayer.crop( x: tileWidth*11,  y: tileHeight*2, tileWidth, tileHeight);

playeridle_up = sheetPlayer.crop( x: tileWidth*10,  y: tileHeight*3, tileWidth, tileHeight);
player_up[0] = sheetPlayer.crop( x: tileWidth*9,  y: tileHeight*3, tileWidth, tileHeight);
player_up[1] = sheetPlayer.crop( x: tileWidth*11,  y: tileHeight*3, tileWidth, tileHeight);

//Potions
health_potion = sheetPotions.crop( x: tileWidth*4, tileHeight, tileWidth, tileHeight);
health_potion_better = sheetPotions.crop( x: 0,  y: tileHeight*7, tileWidth, tileHeight);
max_health_potion = sheetPotions.crop( x: tileWidth*5,  y: tileHeight*8, tileWidth, tileHeight);
mana_potion = sheetPotions.crop( x: 0, tileHeight, tileWidth, tileHeight);
mana_potion_better = sheetPotions.crop( x: tileWidth*2,  y: tileHeight*4, tileWidth, tileHeight);
max_mana_potion = sheetPotions.crop( x: tileWidth*4,  y: 0, tileWidth, tileHeight);
```

The Tile class is a simple class but very important for drawing the maps of the game. It memorizes in an array different static Tile objects with a specific code and image, which is then used by the World class to generate and draw the map. This class also contains methods that verify is the tile is solid or not, with collision purposes.

```java
//static stuff
public static Tile[] tiles = new Tile[256];
public static TileFactory factory = new TileFactory();

public static Tile floorTile = factory.makeTile(TileType.GROUND_1, id: 0);
public static Tile floorTile2 = factory.makeTile(TileType.GROUND_2, id: 1);
public static Tile floorTile3 = factory.makeTile(TileType.GROUND_3, id: 2);
public static Tile wallTile = factory.makeTile(TileType.WALL_1, id: 3);
public static Tile wallTile2 = factory.makeTile(TileType.WALL_2, id: 4);
public static Tile wallTile3 = factory.makeTile(TileType.WALL_3, id: 5);
public static Tile mossywallTile = factory.makeTile(TileType.MOSSY_1, id: 6);
public static Tile mossywallTile2 = factory.makeTile(TileType.MOSSY_2, id: 7);
public static Tile mossywallTile3 = factory.makeTile(TileType.MOSSY_3, id: 8);
public static Tile doorTile = factory.makeTile(TileType.DOOR_1, id: 9);
public static Tile doorTile2 = factory.makeTile(TileType.DOOR_2, id: 10);
public static Tile lavaTile = factory.makeTile(TileType.LAVA_1, id: 11);
public static Tile lavaTile2 = factory.makeTile(TileType.LAVA_2, id: 12);
public static Tile lavaTile3 = factory.makeTile(TileType.LAVA_3, id: 13);

public static final int TILEWIDTH = 64;
public static final int TILEHEIGHT = 64;

//class
protected BufferedImage texture;
protected final int id;
```

The Entity class is another important class, being the base class of every entity in the game, including the player. It contains methods and members (like hitboxes) with purpose in collision with other entities or the “death” of an entity. It gets extended into two main other abstract classes: Moving or Static Entities.

```java
public boolean NoCollisionWithEntities(float xOffset, float yOffset)
{
    for (Entity entity: handler.getWorld().getEntityManager().getEntities())
    {
        if(!entity.equals(this)) {
            if (entity.getHitBox( xOffset: 0f,  yOffset: 0f).intersects(getHitBox(xOffset, yOffset))) //daca hitbox-urile
                return false; //va fi coliziune deci false
        }
    }
    return true;
}
/*! \fn  public Rectangle getHitBox(float xOffset, float yOffset)
            \brief gets "future" hitbox
            */
public Rectangle getHitBox(float xOffset, float yOffset) //get "future" hitbox
{
    return new Rectangle((int)(x + hitbox.x + xOffset), (int)(y+hitbox.y+yOffset), hitbox.width,hitbox.height );
    //basically: x+hitbox.x = exact unde e hitbox-ul. la fel si cu y. xOffset e o "predictie" inainte sa se intample.
}
```

The moving entities have methods for moving, collision with solid tiles, and members such as health, speed with purpose in the way they behave in the game. The Player extends the MovingEntity class, being controllable by the keyboard keys. Every moving entity contains an Animation member, giving life to the entity itself, so it won’t be just a static image.

```
public void takeDamage(int damage)
{
    if (damage>0) //if it's damage that has to be stopped by armor
    {
        if (defense < damage)
            health = health - damage + defense; //defense mitigates damage
        else
            health -= 1; //atleast 1 dmg

        if (health<=0)
        {
            health = 0;
            active = false;
            GetKilled();
        }
    }
    else
    {
        health -= damage;
        if (health>max_health)
        {
            health = max_health;
        }
    }
}
```

The KeyManager class is also important. Without it, the playable character may not be moved by the player, thus making the game not a game. With its help, the player may press different keys with different effects. There's also a MouseManager class, but it's not as important, being used only in Menu State.

```
up = keys[KeyEvent.VK_W];
down = keys[KeyEvent.VK_S];
left = keys[KeyEvent.VK_A];
right = keys[KeyEvent.VK_D];
exit_to_menu = keys[KeyEvent.VK_ESCAPE];
take = keys[KeyEvent.VK_E];
```

The World class is a class used in Game State, and contains the map and map entities. It has an entity manager as member (ArrayList), which modifies, adds or removes entitites when needed from the map. It reads the map from a .txt file as a matrix with coded elements, then using said codes specific for each tile, it raws the map every frame.

```
private void loadWorld(String path)
{
    String file = Utility.loadFileAsString(path);
    String[] tokens = file.split( regex: "\\s+"); //separa nr cu space
    width = Utility.parseInt(tokens[0]);
    height = Utility.parseInt(tokens[1]);
    player_spawnX = Utility.parseInt(tokens[2]);
    player_spawnY = Utility.parseInt(tokens[3]);


    map = new int[width][height];

    for (int y=0; y<height; y++)
    {
        for (int x=0; x<width; x++)
        {
            map[x][y] = Utility.parseInt(tokens[(y*width + x)+4]);
            //vectorul in sine e "inversat" (latimea e prima, inaltimea a doua)
            //dar la desenat merge ok, ptc avem x si y in pozitiile bune
        }
    }
}
```

Handler is a helper class, used in almost every other class. The only special thing about is are its two Game and World members, which may be used when needed by other classes.

The Player class is the way the player interacts with the game. It's a moving entity, movable with WASD keys. It may "kill" (take) static entities (potions), open an inventory in which the "killed" static entities are kept and managed. It has various important stats (variables), such as health, mana, intelligence, defense, which affect the entire game experience (example: health reaches 0, the game ends / game over). It has collisions with tiles and entities, and when too close to a moving entity, it enters the combat state with it. The Player entity is also followed by a GameCamera, which centers it when it's not in the corner or limit of the map.

```
private void Actions()
{
    xMove = 0;
    yMove = 0;
    //very important that they get reset;
    if (!playerInventoryAndStats.isActive()) {
        if (handler.getKeyManager().up)
            yMove = -speed;
        if (handler.getKeyManager().down)
            yMove = speed;
        if (handler.getKeyManager().left)
            xMove = -speed;
        if (handler.getKeyManager().right)
            xMove = speed;
        if (handler.getKeyManager().exit_to_menu) {
            State.setCurrentState(handler.getGame().getMenuState());
        }
        if (handler.getKeyManager().take)
            itemTake();
    }
}
```

Each class has its role in the project, even if that role is to be used by another class.

The classes ImageLoadFunction, StringDrawFunction, Utility and WorldEntities are support classes with helpful methods: load images, create entitites on the map, etc.

The Item class is strictly made to be used by the InventoryAndStats class, appearing after the “death” of static entities, inside a ListArray. These may be consumed by the player with the “e” key, modifying his stats.

Classes from the MenuUI packet have the role to create object buttons in the Menu State, which you may click on for different effects.

There are different Manager classes, with the role of adding the object they manage into an ArrayList. There also are Factory classes, for many different class types.

The SpriteSheet class is used in Assets class, used to “cut” larger images into smaller ones, and the Animations class is used by the extended classes of MovingEntity, containing a BufferedImage array that cycles through its members at a given frequency, with the scope of telling the living entities apart from the static ones.

All diagrams are generated by Doxygen and are inside the Game’s folder.