
STARDUST
CRUSADERS

MEMBERS:

Baltazar, Danilo C.
Calica, Haytchell Reiann C.
De Castro, Angelo Jonin D.
Santiago, Kristoffer B.
Viluan, Patrick Ray S.

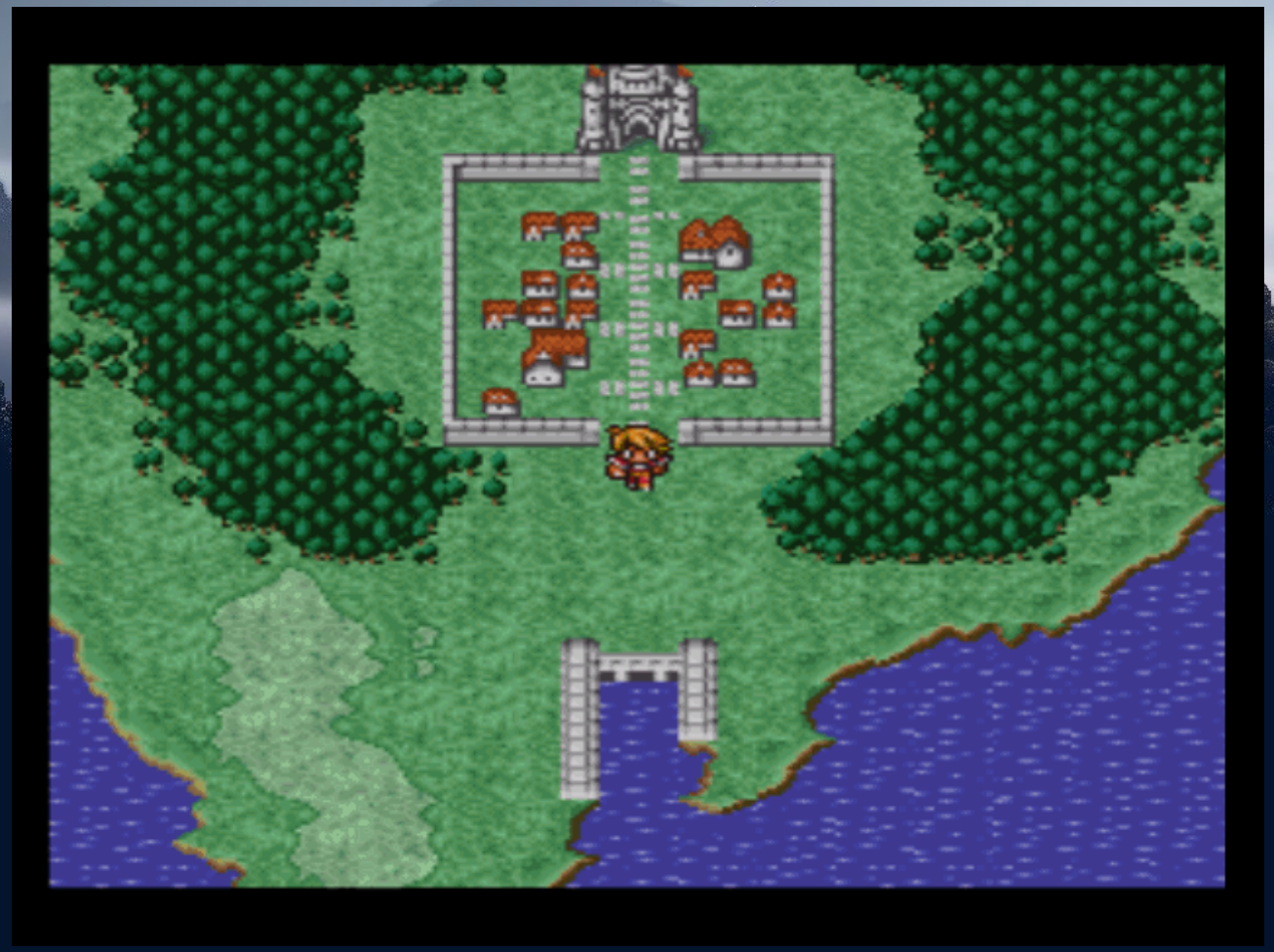
I. Project Description

Story and Concepts Based on:

- Dragon Ball, a 1984 Japanese Manga was serialized and turned into one of the most watched anime until now. Dragon Ball follows the adventures of the protagonist Goku, a strong naive boy who, upon meeting Bulma, sets out to gather the seven wish-granting Dragon Balls

Algorithm and Gameplay Based on:

- Escape Room - also known as an escape game, is a game in which a team of players cooperatively discover clues, solve puzzles, and accomplish tasks in one or more rooms in order to progress and accomplish a specific goal in a limited amount of time.
- In our game collecting item or a certain object open a new path or a new room to explore.



- RPG or Role-Playing Video Game genre began in the mid-1970s, which combined with the ruleset of fantasy wargames. A game in which the participants assume the roles of characters and collaboratively create stories.
- In our game, the player or the character will take the role as a stardust crusaders that is willing to find the magical stardust that can grant any wish.

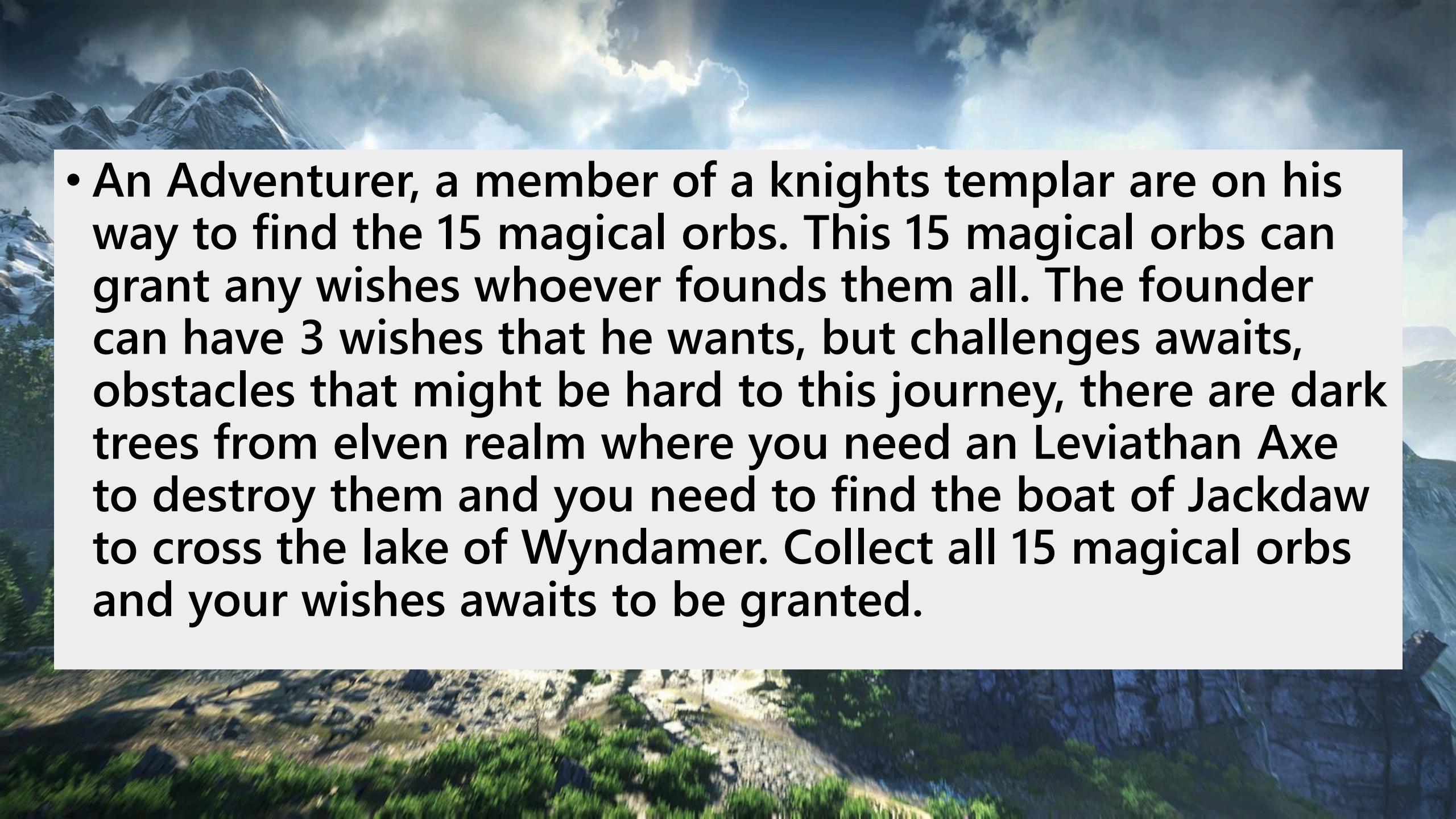
- Open World – is a game mechanic of using a virtual world that the player can explore and approach objectives freely, as opposed to a world with more linear and structured gameplay.

- In our game the player can go back or go further until you find all 15 magical orbs.

- Pokemon Games - the player controls the player character from an overhead perspective and participates in turn-based battles. Throughout the games, the player captures and raises Pokémon for use in battle.

- In our game it is also an overhead perspective which is the same as Pokemon, in our game we also collect or captures 15 magical orbs

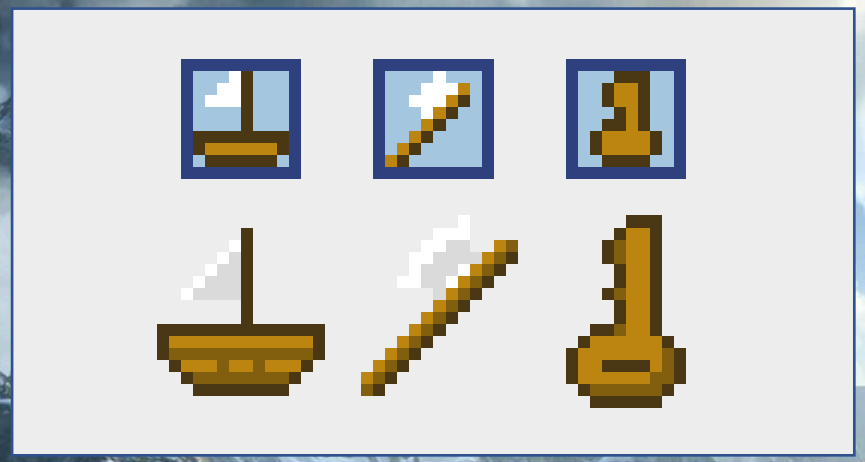
II. Game Description

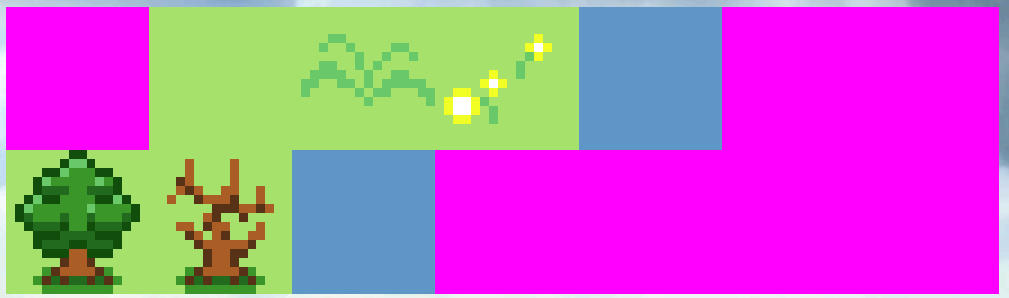


- An Adventurer, a member of a knights templar are on his way to find the 15 magical orbs. This 15 magical orbs can grant any wishes whoever founds them all. The founder can have 3 wishes that he wants, but challenges awaits, obstacles that might be hard to this journey, there are dark trees from elven realm where you need an Leviathan Axe to destroy them and you need to find the boat of Jackdaw to cross the lake of Wyndamer. Collect all 15 magical orbs and your wishes awaits to be granted.

- The member of the knights templar, the adventurer, control this hero and collect all the magical orbs to grant 3 wishes you want.

- The Magical Orb, Where it was created in the realm of the dragon world. Collect them all and win the game and your wishes can come true.





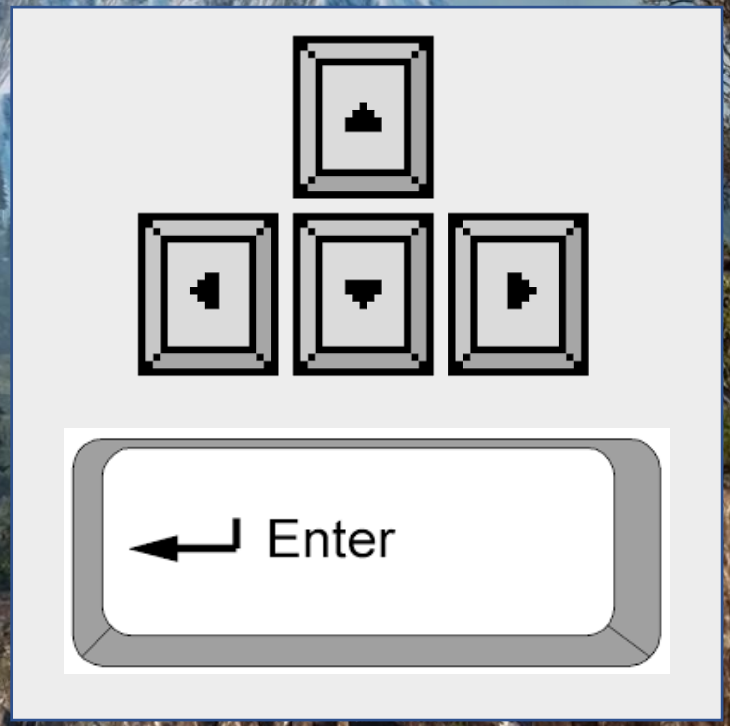
- The Items, Leviathan Axe to destroy the dark trees in your path, the boat of Jackdaw to get across the lake of Wyndamer and get across the Fyke Isle.

- The Tilemap, Fyke Isle is a small island in Velen, located in Lake Wyndamer. A green landscape, green and brown trees, and a blue lake water.

II. Gameplay

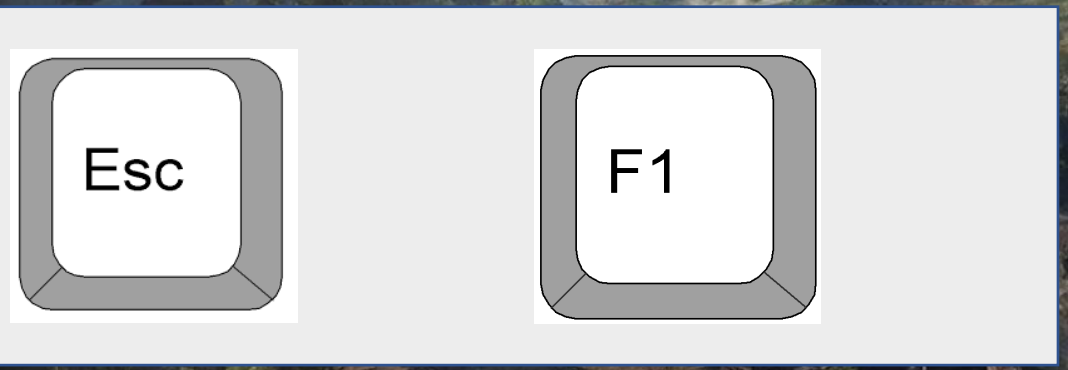
- Arrow Keys: Up, Down, Left, Right Keys: Makes the player move across the map. Also the controls for the menu screen.

- Press the Enter Key whether to start or exit the game at the menu screen.
- Start: will start the game and timer.
- Exit: will exit the game



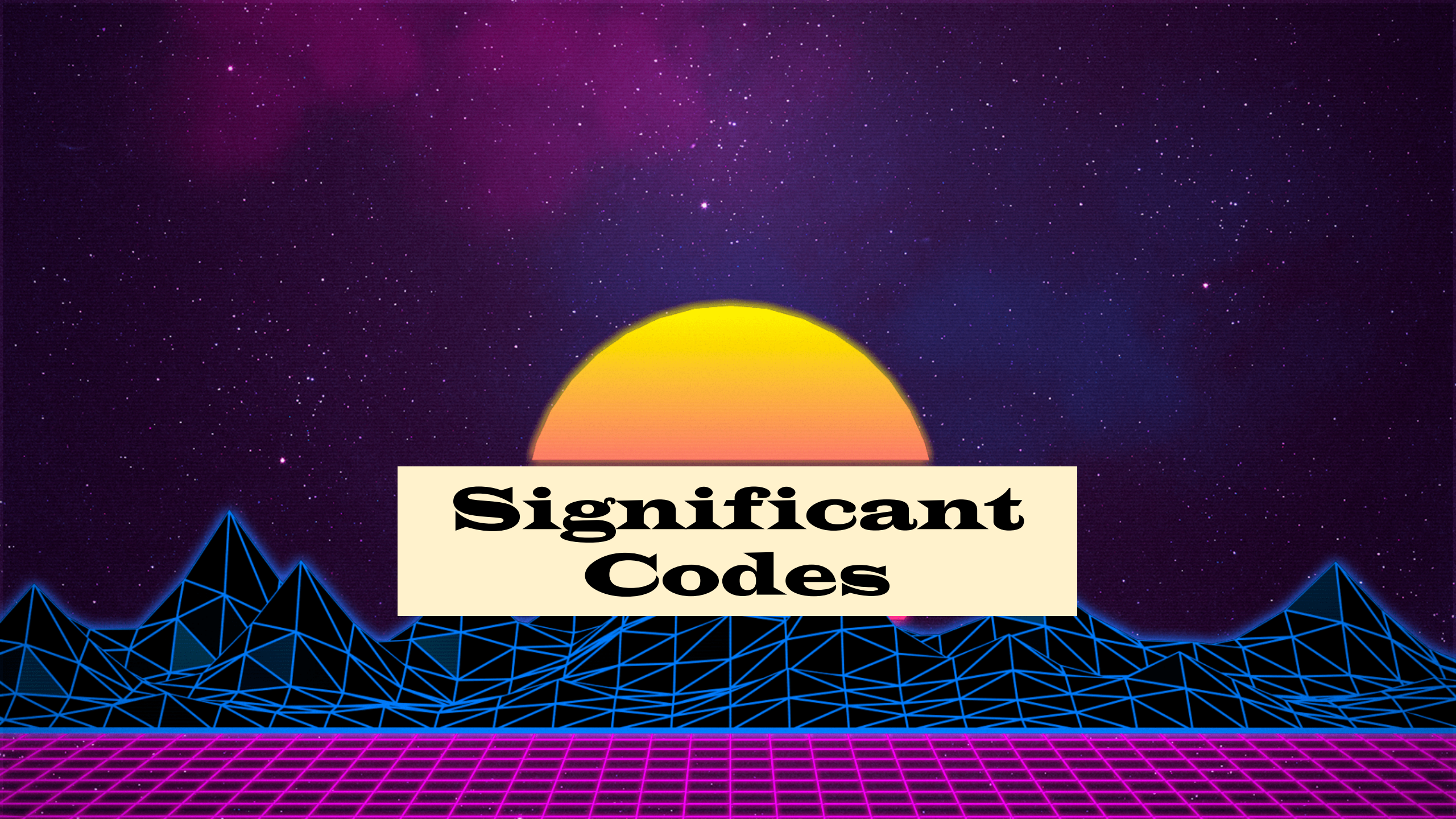
- Space Key: Use your items like the Leviathan axe to cut down the dark trees to clear the path on your journey.

- ESC Key: can pause the game, and it will go to the pause screen. Where it shows the controls of the game which is the arrow keys, spacebar.
- Press the ESC Key again to unpause the game.
- F1 Key: Return To The Main Menu.



- **How to Finish the Game:**
  - **If all 15 Magical Orbs Are Found the game ends, and it will present the time of completing the game, your corresponding rank on the game. The game doesn't have a lose state but only ranking.**

RANKING	TIME
1st God Of Speed	Less Than 2 Minutes
2nd Adventurer	Less Than 3 Minutes
3rd Apprentice	Less Than 4 Minutes
4th Too Slow	4 Minutes or greater


Significant
Codes

Game

```
4  import javax.swing.JFrame;
5
6  public class Game {
7
8      public static void main(String[] args) {
9
10         JFrame window = new JFrame("Stardust Crusaders");
11
12         window.add(new GamePanel());
13
14         window.setResizable(false);
15         window.pack();
16
17         window.setLocationRelativeTo(null);
18         window.setVisible(true);
19         window.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
20
21     }
22
23 }
```

Game Panel

```java
@SuppressWarnings("serial")
public class GamePanel extends JPanel implements Runnable, KeyListener {

	// dimensions
	// HEIGHT is the playing area size
	// HEIGHT2 includes the bottom window
	public static final int WIDTH = 128;
	public static final int HEIGHT = 128;
	public static final int HEIGHT2 = HEIGHT + 16;
	public static final int SCALE = 3;

	// game loop stuff
	private Thread thread;
	private boolean running;
	private final int FPS = 30;
	private final int TARGET_TIME = 1000 / FPS;

	// drawing stuff
	private BufferedImage image;
	private Graphics2D g;

	// game state manager
	private GameStateManager gsm;

	// constructor
	public GamePanel() {
		setPreferredSize(new Dimension(WIDTH * SCALE, HEIGHT2 * SCALE));
		setFocusable(true);
		requestFocus();
	}

```

```java
		}

	}


	private void init() {
		running = true;
		image = new BufferedImage(WIDTH, HEIGHT2, 1);
		g = (Graphics2D) image.getGraphics();
		gsm = new GameStateManager();
	}


	private void update() {
		gsm.update();
		Keys.update();
	}


	private void draw() {
		gsm.draw(g);
	}


	private void drawToScreen() {
		Graphics g2 = getGraphics();
		g2.drawImage(image, 0, 0, WIDTH * SCALE, HEIGHT2 * SCALE, null);
		g2.dispose();
	}


	public void keyTyped(KeyEvent key) {}
	public void keyPressed(KeyEvent key) {
		Keys.keySet(key.getKeyCode(), true);
	}
	public void keyReleased(KeyEvent key) {
		Keys.keySet(key.getKeyCode(), false);
	}

```

```java

	public void addNotify() {
		super.addNotify();
		if(thread == null) {
			addKeyListener(this);
			thread = new Thread(this);
			thread.start();
		}
	}


	public void run() {

		init();

		long start;
		long elapsed;
		long wait;


		while(running) {

			start = System.nanoTime();

			update();
			draw();
			drawToScreen();

			elapsed = System.nanoTime() - start;

			wait = TARGET_TIME - elapsed / 1000000;
			if(wait < 0) wait = TARGET_TIME;

			try {
				Thread.sleep(wait);
			}
			catch(Exception e) {
				e.printStackTrace();
			}

```

Content

```
11
12  public class Content {
13
14      public static BufferedImage[][] MENUBG = load("/HUD/menuscreen.gif", 128, 144);
15      public static BufferedImage[][] BAR = load("/HUD/bar.gif", 128, 16);
16
17      public static BufferedImage[][] PLAYER = load("/Sprites/playersprites.gif", 16, 16);
18      public static BufferedImage[][] DIAMOND = load("/Sprites/diamond.gif", 16, 16);
19      public static BufferedImage[][] SPARKLE = load("/Sprites/sparkle.gif", 16, 16);
20      public static BufferedImage[][] ITEMS = load("/Sprites/items.gif", 16, 16);
21
22      public static BufferedImage[][] font = load("/HUD/font.gif", 8, 8);
23
24      public static BufferedImage[][] load(String s, int w, int h) {
25          BufferedImage[][] ret;
26          try {
27              BufferedImage spritesheet = ImageIO.read(Content.class.getResourceAsStream(s));
28              int width = spritesheet.getWidth() / w;
29              int height = spritesheet.getHeight() / h;
30              ret = new BufferedImage[height][width];
31              for(int i = 0; i < height; i++) {
32                  for(int j = 0; j < width; j++) {
33                      ret[i][j] = spritesheet.getSubimage(j * w, i * h, w, h);
34                  }
35              }
36              return ret;
37          }
          catch(Exception e) {
              e.printStackTrace();
40            System.out.println("Error loading graphics.");
41            System.exit(0);
42        }
43        return null;
44    }
45
```

```
45
46      public static void drawString(Graphics2D g, String s, int x, int y) {
47          s = s.toUpperCase();
48          for(int i = 0; i < s.length(); i++) {
49              char c = s.charAt(i);
50              if(c == 47) c = 36; // slash
51              if(c == 58) c = 37; // colon
52              if(c == 32) c = 38; // space
53              if(c >= 65 && c <= 90) c -= 65; // letters
54              if(c >= 48 && c <= 57) c -= 22; // numbers
55              int row = c / font[0].length;
56              int col = c % font[0].length;
57              g.drawImage(font[row][col], x + 8 * i, y, null);
58          }
59      }
60
61  }
62
```

Data

```
4   package com.neet.DiamondHunter.Manager;
5
6   public class Data {
7
8           public static long time;
9
10          public static void setTime(long l) { time = l; }
11          public static long getTime() { return time; }
12
13  }
14
```

Game State Manager

```
19  public class GameStateManager {
20
21      private boolean paused;
22      private PauseState pauseState;
23
24      private GameState[] gameStates;
25      private int currentState;
26      private int previousState;
27
28      public static final int NUM_STATES = 4;
29      public static final int INTRO = 0;
30      public static final int MENU = 1;
31      public static final int PLAY = 2;
32      public static final int GAMEOVER = 3;
33
34      public GameStateManager() {
35
36          JukeBox.init();
37
38          paused = false;
39          pauseState = new PauseState(this);
40
41          gameStates = new GameState[NUM_STATES];
42          setState(INTRO);
43
44      }
45
46      public void setState(int i) {
47          previousState = currentState;
48          unloadState(previousState);
49          currentState = i;
50          if(i == INTRO) {
51              gameStates[i] = new IntroState(this);
52              gameStates[i].init();
53          }
54          else if(i == MENU) {
```

```
60              gameStates[i].init();
61          }
62          else if(i == GAMEOVER) {
63              gameStates[i] = new GameOverState(this);
64              gameStates[i].init();
65          }
66      }
67
68      public void unloadState(int i) {
69          gameStates[i] = null;
70      }
71
72      public void setPaused(boolean b) {
73          paused = b;
74      }
75
76      public void update() {
77          if(paused) {
78              pauseState.update();
79          }
80          else if(gameStates[currentState] != null) {
81              gameStates[currentState].update();
82          }
83      }
84
85      public void draw(Graphics2D g) {
86          if(paused) {
87              pauseState.draw(g);
88          }
89          else if(gameStates[currentState] != null) {
90              gameStates[currentState].draw(g);
91          }
92      }
93
94  }
```

JukeBox

```java
import javax.sound.sampled.AudioInputStream;
import javax.sound.sampled.AudioSystem;
import javax.sound.sampled.Clip;
import javax.sound.sampled.FloatControl;

public class JukeBox {

    private static HashMap<String, Clip> clips;
    private static int gap;


    public static void init() {
        clips = new HashMap<String, Clip>();
        gap = 0;
    }


    public static void load(String s, String n) {
        if(clips.get(n) != null) return;
        Clip clip;
        try {
            InputStream in = JukeBox.class.getResourceAsStream(s);
            InputStream bin = new BufferedInputStream(in);
            AudioInputStream ais =
                    AudioSystem.getAudioInputStream(bin);
            AudioFormat baseFormat = ais.getFormat();
            AudioFormat decodeFormat = new AudioFormat(
                    AudioFormat.Encoding.PCM_SIGNED,
                    baseFormat.getSampleRate(),
                    16,
                    baseFormat.getChannels(),
                    baseFormat.getChannels() * 2,
                    baseFormat.getSampleRate(),
                    false
            );
            AudioInputStream dais = AudioSystem.getAudioInputStream
            clip = AudioSystem.getClip();
            clip.open(dais);
```

```java
            }
            catch(Exception e) {
                e.printStackTrace();
            }
        }

        public static void play(String s) {
            play(s, gap);
        }

        public static void play(String s, int i) {
            Clip c = clips.get(s);
            if(c == null) return;
            if(c.isRunning()) c.stop();
            c.setFramePosition(i);
            while(!c.isRunning()) c.start();
        }

        public static void stop(String s) {
            if(clips.get(s) == null) return;
            if(clips.get(s).isRunning()) clips.get(s).stop();
        }

        public static void resume(String s) {
            if(clips.get(s).isRunning()) return;
            clips.get(s).start();
        }

        public static void resumeLoop(String s) {
            Clip c = clips.get(s);
            if(c == null) return;
            c.loop(Clip.LOOP_CONTINUOUSLY);
        }

        public static void loop(String s) {
            loop(s, gap, gap, clips.get(s).getFrameLength() - 1);
```

```java
        }

        public static void loop(String s, int frame, int start, int end) {
            Clip c = clips.get(s);
            if(c == null) return;
            if(c.isRunning()) c.stop();
            c.setLoopPoints(start, end);
            c.setFramePosition(frame);
            c.loop(Clip.LOOP_CONTINUOUSLY);
        }

        public static void setPosition(String s, int frame) {
            clips.get(s).setFramePosition(frame);
        }

        public static int getFrames(String s) { return clips.get(s).getFrameLength(); }
        public static int getPosition(String s) { return clips.get(s).getFramePosition(); }

        public static void close(String s) {
            stop(s);
            clips.get(s).close();
        }

        public static void setVolume(String s, float f) {
            Clip c = clips.get(s);
            if(c == null) return;
            FloatControl vol = (FloatControl) c.getControl(FloatControl.Type.MASTER_GAIN);
            vol.setValue(f);
        }

        public static boolean isPlaying(String s) {
            Clip c = clips.get(s);
            if(c == null) return false;
            return c.isRunning();
        }
```

Keys

```
15  public class Keys {
16
17      public static final int NUM_KEYS = 8;
18
19      public static boolean keyState[] = new boolean[NUM_KEYS];
20      public static boolean prevKeyState[] = new boolean[NUM_KEYS];
21
22      public static int UP = 0;
23      public static int LEFT = 1;
24      public static int DOWN = 2;
25      public static int RIGHT = 3;
26      public static int SPACE = 4;
27      public static int ENTER = 5;
28      public static int ESCAPE = 6;
29      public static int F1 = 7;
30
31      public static void keySet(int i, boolean b) {
            if(i == KeyEvent.VK_UP) keyState[UP] = b;
33          else if(i == KeyEvent.VK_LEFT) keyState[LEFT] = b;
34          else if(i == KeyEvent.VK_DOWN) keyState[DOWN] = b;
35          else if(i == KeyEvent.VK_RIGHT) keyState[RIGHT] = b;
36          else if(i == KeyEvent.VK_SPACE) keyState[SPACE] = b;
37          else if(i == KeyEvent.VK_ENTER) keyState[ENTER] = b;
38          else if(i == KeyEvent.VK_ESCAPE) keyState[ESCAPE] = b;
39          else if(i == KeyEvent.VK_F1) keyState[F1] = b;
40      }
41
42      public static void update() {
            for(int i = 0; i < NUM_KEYS; i++) {
44              prevKeyState[i] = keyState[i];
45          }
46      }
47
48      public static boolean isPressed(int i) {
49          return keyState[i] && !prevKeyState[i];
```

```
38          else if(i == KeyEvent.VK_ESCAPE) keyState[ESCAPE] = b;
39          else if(i == KeyEvent.VK_F1) keyState[F1] = b;
40      }
41
42      public static void update() {
            for(int i = 0; i < NUM_KEYS; i++) {
44              prevKeyState[i] = keyState[i];
45          }
46      }
47
48      public static boolean isPressed(int i) {
49          return keyState[i] && !prevKeyState[i];
50      }
51
52      public static boolean isDown(int i) {
53          return keyState[i];
54      }
55
56      public static boolean anyKeyDown() {
57          for(int i = 0; i < NUM_KEYS; i++) {
58              if(keyState[i]) return true;
59          }
60          return false;
61      }
62
63      public static boolean anyKeyPress() {
64          for(int i = 0; i < NUM_KEYS; i++) {
65              if(keyState[i] && !prevKeyState[i]) return true;
66          }
67          return false;
68      }
69
70  }
71
```

Game Over State

```java
import java.awt.Color;
import java.awt.Graphics2D;

import com.neet.DiamondHunter.Main.GamePanel;
import com.neet.DiamondHunter.Manager.Content;
import com.neet.DiamondHunter.Manager.Data;
import com.neet.DiamondHunter.Manager.GameStateManager;
import com.neet.DiamondHunter.Manager.JukeBox;
import com.neet.DiamondHunter.Manager.Keys;

public class GameOverState extends GameState {

	private Color color;

	private int rank;
	private long ticks;

	public GameOverState(GameStateManager gsm) {
		super(gsm);
	}

	public void init() {
		color = new Color(164, 198, 222);
		ticks = Data.getTime();
		if(ticks < 3600) rank = 1;
		else if(ticks < 5400) rank = 2;
		else if(ticks < 7200) rank = 3;
		else rank = 4;
	}

	public void update() {}

	public void draw(Graphics2D g) {

		g.setColor(color);
		g.fillRect(0, 0, GamePanel.WIDTH, GamePanel.HEIGHT2);

		Content.drawString(g, "finish time", 20, 20);

		int minutes = (int) (ticks / 1800);
		int seconds = (int) ((ticks / 30) % 60);
		if(minutes < 10) {
			if(seconds < 10) Content.drawString(g, "0" + minutes + ":0" + seconds, 44, 32);
			else Content.drawString(g, "0" + minutes + ":" + seconds, 44, 32);
		}
		else {
			if(seconds < 10) Content.drawString(g, minutes + ":0" + seconds, 44, 32);
			else Content.drawString(g, minutes + ":" + seconds, 44, 22);
		}
```

```java
		int minutes = (int) (ticks / 1800);
		int seconds = (int) ((ticks / 30) % 60);
		if(minutes < 10) {
			if(seconds < 10) Content.drawString(g, "0" + minutes + ":0" + seconds, 44, 32);
			else Content.drawString(g, "0" + minutes + ":" + seconds, 44, 32);
		}
		else {
			if(seconds < 10) Content.drawString(g, minutes + ":0" + seconds, 44, 32);
			else Content.drawString(g, minutes + ":" + seconds, 44, 22);
		}

		Content.drawString(g, "rank", 48, 45);
		if(rank == 1) Content.drawString(g, "God of Speed", 20, 58);
		else if(rank == 2) Content.drawString(g, "adventurer", 24, 58);
		else if(rank == 3) Content.drawString(g, "apprentice", 32, 58);
		else if(rank == 4) Content.drawString(g, "too slow", 8, 58);

		Content.drawString(g, "Your Three", 15, 75);
		Content.drawString(g, "Wishes May", 15, 95);
		Content.drawString(g, "Be Granted", 15, 115);

	}

	public void handleInput() {
		if(Keys.isPressed(Keys.ENTER)) {
			gsm.setState(GameStateManager.MENU);
			JukeBox.play("collect");
		}
	}

}
```

Game State

```
public abstract class GameState {

    protected GameStateManager gsm;

    public GameState(GameStateManager gsm) {
        this.gsm = gsm;
    }

    public abstract void init();
    public abstract void update();
    public abstract void draw(Graphics2D g);
    public abstract void handleInput();

}
```

Intro State

```
13  import com.neet.DiamondHunter.Manager.Keys;
14
15  public class IntroState extends GameState {
16
17      private BufferedImage logo;
18
19      private int alpha;
20      private int ticks;
21
22      private final int FADE_IN = 60;
23      private final int LENGTH = 60;
24      private final int FADE_OUT = 60;
25
26      public IntroState(GameStateManager gsm) {
27          super(gsm);
28      }
29
30      public void init() {
31          ticks = 0;
32          try {
33              logo = ImageIO.read(getClass().getResourceAsStream("/Logo/logo.gif"));
34          }
35          catch(Exception e) {
36              e.printStackTrace();
37          }
38      }
39
40      public void update() {
41          handleInput();
42          ticks++;
43          if(ticks < FADE_IN) {
44              alpha = (int) (255 - 255 * (1.0 * ticks / FADE_IN));
45              if(alpha < 0) alpha = 0;
46          }
47          if(ticks > FADE_IN + LENGTH) {
48              alpha = (int) (255 * (1.0 * ticks - FADE_IN - LENGTH) / FADE_OUT);
49              if(alpha > 255) alpha = 255;
50          }
51          if(ticks > FADE_IN + LENGTH + FADE_OUT) {
52              gsm.setState(GameStateManager.MENU);
53          }
54      }
55
56      public void draw(Graphics2D g) {
57          g.setColor(Color.WHITE);
58          g.fillRect(0, 0, GamePanel.WIDTH, GamePanel.HEIGHT2);
59          g.drawImage(logo, 0, 0, GamePanel.WIDTH, GamePanel.HEIGHT2, null);
60          g.setColor(new Color(0, 0, 0, alpha));
61          g.fillRect(0, 0, GamePanel.WIDTH, GamePanel.HEIGHT2);
62      }
63
64      public void handleInput() {
65          if(Keys.isPressed(Keys.ENTER)) {
66              gsm.setState(GameStateManager.MENU);
67          }
68      }
69
70  }
```

Menu State

```
13  public class MenuState extends GameState {
14
15      private BufferedImage bg;
16      private BufferedImage diamond;
17
18      private int currentOption = 0;
19      private String[] options = {
20              "START",
21              "QUIT"
22      };
23
24      public MenuState(GameStateManager gsm) {
25          super(gsm);
26      }
27
28      public void init() {
29          bg = Content.MENUBG[0][0];
30          diamond = Content.DIAMOND[0][0];
31          JukeBox.load("/SFX/collect.wav", "collect");
32          JukeBox.load("/SFX/menuoption.wav", "menuoption");
33      }
34
35      public void update() {
36          handleInput();
37      }
38
39      public void draw(Graphics2D g) {
40
41          g.drawImage(bg, 0, 0, null);
42
43          Content.drawString(g, options[0], 44, 90);
44          Content.drawString(g, options[1], 48, 100);
```

```
43          Content.drawString(g, options[0], 44, 90);
44          Content.drawString(g, options[1], 48, 100);
45
46          if(currentOption == 0) g.drawImage(diamond, 25, 86, null);
47          else if(currentOption == 1) g.drawImage(diamond, 25, 96, null);
48
49      }
50
51      public void handleInput() {
52          if(Keys.isPressed(Keys.DOWN) && currentOption < options.length - 1) {
53              JukeBox.play("menuoption");
54              currentOption++;
55          }
56          if(Keys.isPressed(Keys.UP) && currentOption > 0) {
57              JukeBox.play("menuoption");
58              currentOption--;
59          }
60          if(Keys.isPressed(Keys.ENTER)) {
61              JukeBox.play("collect");
62              selectOption();
63          }
64      }
65
66      private void selectOption() {
67          if(currentOption == 0) {
68              gsm.setState(GameStateManager.PLAY);
69          }
70          if(currentOption == 1) {
71              System.exit(0);
72          }
73      }
74
```

Pause State

```
15      public PauseState(GameStateManager gsm) {
16          super(gsm);
17      }
18
19      public void init() {}
20
21      public void update() {
22          handleInput();
23      }
24
25      public void draw(Graphics2D g) {
26
27          Content.drawString(g, "paused", 40, 30);
28
29          Content.drawString(g, "arrow", 12, 76);
30          Content.drawString(g, "keys", 16, 84);
31          Content.drawString(g, ": move", 52, 80);
32
33          Content.drawString(g, "space", 12, 96);
34          Content.drawString(g, ": action", 52, 96);
35
36          Content.drawString(g, "F1:", 36, 112);
37          Content.drawString(g, "return", 68, 108);
38          Content.drawString(g, "to menu", 68, 116);
39
40      }
41      public void handleInput() {
42          if(Keys.isPressed(Keys.ESCAPE)) {
43              gsm.setPaused(false);
44              JukeBox.resumeLoop("music1");
45          }
46          if(Keys.isPressed(Keys.F1)) {
47              gsm.setPaused(false);
48              gsm.setState(GameStateManager.MENU);
49          }
50      }
```

Play State

```
public class PlayState extends GameState {

        private Player player;

        private TileMap tileMap;

        private ArrayList<Diamond> diamonds;

        private ArrayList<Item> items;

        private ArrayList<Sparkle> sparkles;

        private int xsector;
        private int ysector;
        private int sectorSize;

        private Hud hud;

        private boolean blockInput;
        private boolean eventStart;
        private boolean eventFinish;
        private int eventTick;
```

```
        player.setTilePosition(17, 17);
        player.setTotalDiamonds(diamonds.size());


        sectorSize = GamePanel.WIDTH;
        xsector = player.getx() / sectorSize;
        ysector = player.gety() / sectorSize;
        tileMap.setPositionImmediately(-xsector * sectorSize, -ysector * sectorSize);


        hud = new Hud(player, diamonds);

        JukeBox.load("/Music/bgmusic.mp3", "music1");
        JukeBox.setVolume("music1", -10);
        JukeBox.loop("music1", 1000, 1000, JukeBox.getFrames("music1") - 1000);
        JukeBox.load("/Music/finish.mp3", "finish");
        JukeBox.setVolume("finish", -10);


        JukeBox.load("/SFX/collect.wav", "collect");
        JukeBox.load("/SFX/mapmove.wav", "mapmove");
        JukeBox.load("/SFX/tilechange.wav", "tilechange");
        JukeBox.load("/SFX/splash.wav", "splash");


        boxes = new ArrayList<Rectangle>();
        eventStart = true;
        eventStart();
    }
```

```
    private void populateItems() {

        Item item;

        item = new Item(tileMap);
        item.setType(Item.AXE);
        item.setTilePosition(26, 37);
        items.add(item);

        item = new Item(tileMap);
        item.setType(Item.BOAT);
        item.setTilePosition(12, 4);
        items.add(item);

    }

    public void update() {


        handleInput();


        if(eventStart) eventStart();
        if(eventFinish) eventFinish();

        if(player.numDiamonds() == player.getTotalDiamonds()) {
            eventFinish = blockInput = true;
        }
```

```
        }


        for(int i = 0; i < sparkles.size(); i++) {
            Sparkle s = sparkles.get(i);
            s.update();
            if(s.shouldRemove()) {
                sparkles.remove(i);
                i--;
            }
        }


        for(int i = 0; i < items.size(); i++) {
            Item item = items.get(i);
            if(player.intersects(item)) {
                items.remove(i);
                i--;
                item.collected(player);
                JukeBox.play("collect");
                Sparkle s = new Sparkle(tileMap);
                s.setPosition(item.getx(), item.gety());
                sparkles.add(s);
            }
        }

    }

    public void draw(Graphics2D g) {


        tileMap.draw(g);


        player.draw(g);
```

```
    public void handleInput() {
        if(Keys.isPressed(Keys.ESCAPE)) {
            JukeBox.stop("music1");
            gsm.setPaused(true);
        }
        if(blockInput) return;
        if(Keys.isDown(Keys.LEFT)) player.setLeft();
        if(Keys.isDown(Keys.RIGHT)) player.setRight();
        if(Keys.isDown(Keys.UP)) player.setUp();
        if(Keys.isDown(Keys.DOWN)) player.setDown();
        if(Keys.isPressed(Keys.SPACE)) player.setAction();
    }

    private void eventStart() {
        eventTick++;
        if(eventTick == 1) {
            boxes.clear();
            for(int i = 0; i < 9; i++) {
                boxes.add(new Rectangle(0, i * 16, GamePanel.WIDTH, 16));
            }
        }
        if(eventTick > 1 && eventTick < 32) {
            for(int i = 0; i < boxes.size(); i++) {
                Rectangle r = boxes.get(i);
                if(i % 2 == 0) {
                    r.x -= 4;
                }
                else {
                    r.x += 4;
                }
            }
        }
        if(eventTick == 33) {
            boxes.clear();
            eventStart = false;
```

Animation

```java
public class Animation {

    private BufferedImage[] frames;
    private int currentFrame;
    private int numFrames;

    private int count;
    private int delay;

    private int timesPlayed;

    public Animation() {
        timesPlayed = 0;
    }

    public void setFrames(BufferedImage[] frames) {
        this.frames = frames;
        currentFrame = 0;
        count = 0;
        timesPlayed = 0;
        delay = 2;
        numFrames = frames.length;
    }

    public void setDelay(int i) { delay = i; }
    public void setFrame(int i) { currentFrame = i; }
    public void setNumFrames(int i) { numFrames = i; }

    public void update() {

        if(delay == -1) return;

        count++;

        if(count == delay) {
            currentFrame++;
            count = 0;
        }
        if(currentFrame == numFrames) {
            currentFrame = 0;
            timesPlayed++;
        }

    }

    public int getFrame() { return currentFrame; }
    public int getCount() { return count; }
    public BufferedImage getImage() { return frames[currentFrame]; }
    public boolean hasPlayedOnce() { return timesPlayed > 0; }
    public boolean hasPlayed(int i) { return timesPlayed == i; }

}
```

Diamond/Magical Orbs

```
14
15  public class Diamond extends Entity {
16
17      BufferedImage[] sprites;
18
19      private ArrayList<int[]> tileChanges;
20
21      public Diamond(TileMap tm) {
22
23          super(tm);
24
25          width = 16;
26          height = 16;
27          cwidth = 12;
28          cheight = 12;
29
30          sprites = Content.DIAMOND[0];
31          animation.setFrames(sprites);
32          animation.setDelay(10);
33
34          tileChanges = new ArrayList<int[]>();
35
36      }
37
38      public void addChange(int[] i) {
39          tileChanges.add(i);
40      }
41      public ArrayList<int[]> getChanges() {
42          return tileChanges;
43      }
44
45      public void update() {
46          animation.update();
47      }
48
49      public void draw(Graphics2D g) {
50          super.draw(g);
51      }
52
53  }
54
```

Entity

```
            xdest = x;
            ydest = y;
        }

        public void setLeft() {
            if(moving) return;
            left = true;
            moving = validateNextPosition();
        }
        public void setRight() {
            if(moving) return;
            right = true;
            moving = validateNextPosition();
        }
        public void setUp() {
            if(moving) return;
            up = true;
            moving = validateNextPosition();
        }
        public void setDown() {
            if(moving) return;
            down = true;
            moving = validateNextPosition();
        }

        public boolean intersects(Entity e) {
            return getRectangle().intersects(e.getRectangle());
        }

        public Rectangle getRectangle() {
            return new Rectangle(x, y, cwidth, cheight);
        }
```

```
public abstract class Entity {

    // dimensions
    protected int width;
    protected int height;
    protected int cwidth;
    protected int cheight;

    // position
    protected int x;
    protected int y;
    protected int xdest;
    protected int ydest;
    protected int rowTile;
    protected int colTile;

    // movement
    protected boolean moving;
    protected boolean left;
    protected boolean right;
    protected boolean up;
    protected boolean down;

    // attributes
    protected int moveSpeed;

    // tilemap
    protected TileMap tileMap;
    protected int tileSize;
    protected int xmap;
    protected int ymap;

```

```
    public void getNextPosition() {

        if(left && x > xdest) x -= moveSpeed;
        else left = false;
        if(left && x < xdest) x = xdest;

        if(right && x < xdest) x += moveSpeed;
        else right = false;
        if(right && x > xdest) x = xdest;

        if(up && y > ydest) y -= moveSpeed;
        else up = false;
        if(up && y < ydest) y = ydest;

        if(down && y < ydest) y += moveSpeed;
        else down = false;
        if(down && y > ydest) y = ydest;

    }

    public void update() {


        if(moving) getNextPosition();


        if(x == xdest && y == ydest) {
            left = right = up = down = moving = false;
            rowTile = y / tileSize;
            colTile = x / tileSize;
        }
```

```
        if(left) {
            if(colTile == 0 || tileMap.getType(rowTile, colTile - 1) == Tile.BLOCKED) {
                return false;
            }
            else {
                xdest = x - tileSize;
            }
        }
        if(right) {
            if(colTile == tileMap.getNumCols() || tileMap.getType(rowTile, colTile + 1) == Tile.BLOCKED) {
                return false;
            }
            else {
                xdest = x + tileSize;
            }
        }
        if(up) {
            if(rowTile == 0 || tileMap.getType(rowTile - 1, colTile) == Tile.BLOCKED) {
                return false;
            }
            else {
                ydest = y - tileSize;
            }
        }
        if(down) {
            if(rowTile == tileMap.getNumRows() - 1 || tileMap.getType(rowTile + 1, colTile) == Tile.BLOCKED) {
                return false;
            }
            else {
                ydest = y + tileSize;
            }
```

```
    protected Animation animation;
    protected int currentAnimation;

    public Entity(TileMap tm) {
        tileMap = tm;
        tileSize = tileMap.getTileSize();
        animation = new Animation();
    }

    public int getx() { return x; }
    public int gety() { return y; }
    public int getRow() { return rowTile; }
    public int getCol() { return colTile; }

    public void setPosition(int i1, int i2) {
        x = i1;
        y = i2;
        xdest = x;
        ydest = y;
    }
    public void setMapPosition() {
        xmap = tileMap.getx();
        ymap = tileMap.gety();
    }
    public void setTilePosition(int i1, int i2) {
        y = i1 * tileSize + tileSize / 2;
        x = i2 * tileSize + tileSize / 2;
        xdest = x;
        ydest = y;
    }

    public void setLeft() {
```

```
    public void update() {


        if(moving) getNextPosition();


        if(x == xdest && y == ydest) {
            left = right = up = down = moving = false;
            rowTile = y / tileSize;
            colTile = x / tileSize;
        }


        animation.update();

    }

    public void draw(Graphics2D g) {
        setMapPosition();
        g.drawImage(
            animation.getImage(),
            x + xmap - width / 2,
            y + ymap - height / 2,
            null
        );
    }

}
```

Item

```
            cwidth = cheight = 12;
        }

        public void setType(int i) {
            type = i;
            if(type == BOAT) {
                sprite = Content.ITEMS[1][0];
            }
            else if(type == AXE) {
                sprite = Content.ITEMS[1][1];
            }
        }

        public void collected(Player p) {
            if(type == BOAT) {
                p.gotBoat();
            }
            if(type == AXE) {
                p.gotAxe();
            }
        }

        public void draw(Graphics2D g) {
            setMapPosition();
            g.drawImage(sprite, x + xmap - width / 2, y + ymap - height / 2, null);
        }

    }

```

```
public class Item extends Entity{

        private BufferedImage sprite;
        private int type;
        public static final int BOAT = 0;
        public static final int AXE = 1;

        public Item(TileMap tm) {
            super(tm);
            type = -1;
            width = height = 16;
            cwidth = cheight = 12;
        }

        public void setType(int i) {
            type = i;
            if(type == BOAT) {
                sprite = Content.ITEMS[1][0];
            }
            else if(type == AXE) {
                sprite = Content.ITEMS[1][1];
            }
        }

        public void collected(Player p) {
            if(type == BOAT) {
                p.gotBoat();
            }
            if(type == AXE) {
                p.gotAxe();
            }
```

Player

```
public class Player extends Entity {


        private BufferedImage[] downSprites;
        private BufferedImage[] leftSprites;
        private BufferedImage[] rightSprites;
        private BufferedImage[] upSprites;
        private BufferedImage[] downBoatSprites;
        private BufferedImage[] leftBoatSprites;
        private BufferedImage[] rightBoatSprites;
        private BufferedImage[] upBoatSprites;


        private final int DOWN = 0;
        private final int LEFT = 1;
        private final int RIGHT = 2;
        private final int UP = 3;
        private final int DOWNBOAT = 4;
        private final int LEFTBOAT = 5;
        private final int RIGHTBOAT = 6;
        private final int UPBOAT = 7;

        private int numDiamonds;
        private int totalDiamonds;
        private boolean hasBoat;
        private boolean hasAxe;
        private boolean onWater;
        private long ticks;

        public Player(TileMap tm) {
```

```
        public Player(TileMap tm) {

                super(tm);

                width = 16;
                height = 16;
                cwidth = 12;
                cheight = 12;

                moveSpeed = 2;

                numDiamonds = 0;

                downSprites = Content.PLAYER[0];
                leftSprites = Content.PLAYER[1];
                rightSprites = Content.PLAYER[2];
                upSprites = Content.PLAYER[3];
                downBoatSprites = Content.PLAYER[4];
                leftBoatSprites = Content.PLAYER[5];
                rightBoatSprites = Content.PLAYER[6];
                upBoatSprites = Content.PLAYER[7];

                animation.setFrames(downSprites);
                animation.setDelay(10);

        }

        private void setAnimation(int i, BufferedImage[] bi, int d) {
                currentAnimation = i;
                animation.setFrames(bi);
                animation.setDelay(d);
        }
```

```
        private void setAnimation(int i, BufferedImage[] bi, int d) {
                currentAnimation = i;
                animation.setFrames(bi);
                animation.setDelay(d);
        }

        public void collectedDiamond() { numDiamonds++; }
        public int numDiamonds() { return numDiamonds; }
        public int getTotalDiamonds() { return totalDiamonds; }
        public void setTotalDiamonds(int i) { totalDiamonds = i; }

        public void gotBoat() { hasBoat = true; tileMap.replace(22, 4); }
        public void gotAxe() { hasAxe = true; }
        public boolean hasBoat() { return hasBoat; }
        public boolean hasAxe() { return hasAxe; }


        public long getTicks() { return ticks; }


        public void setDown() {
                super.setDown();
        }
        public void setLeft() {
                super.setLeft();
        }
        public void setRight() {
                super.setRight();
        }
        public void setUp() {
                super.setUp();
        }
```

```
public void setAction() {
        if(hasAxe) {
                if(currentAnimation == UP && tileMap.getIndex(rowTile - 1, colTile) == 21) {
                        tileMap.setTile(rowTile - 1, colTile, 1);
                        JukeBox.play("tilechange");
                }
                if(currentAnimation == DOWN && tileMap.getIndex(rowTile + 1, colTile) == 21) {
                        tileMap.setTile(rowTile + 1, colTile, 1);
                        JukeBox.play("tilechange");
                }
                if(currentAnimation == LEFT && tileMap.getIndex(rowTile, colTile - 1) == 21) {
                        tileMap.setTile(rowTile, colTile - 1, 1);
                        JukeBox.play("tilechange");
                }
                if(currentAnimation == RIGHT && tileMap.getIndex(rowTile, colTile + 1) == 21) {
                        tileMap.setTile(rowTile, colTile + 1, 1);
                        JukeBox.play("tilechange");
                }
        }
}

public void update() {

        ticks++;


        boolean current = onWater;
        if(tileMap.getIndex(ydest / tileSize, xdest / tileSize) == 4) {
                onWater = true;
        }
        else {
```

```
                        if(onWater && currentAnimation != LEFTBOAT) {
                                setAnimation(LEFTBOAT, leftBoatSprites, 10);
                        }
                        else if(!onWater && currentAnimation != LEFT) {
                                setAnimation(LEFT, leftSprites, 10);
                        }
                }
                if(right) {
                        if(onWater && currentAnimation != RIGHTBOAT) {
                                setAnimation(RIGHTBOAT, rightBoatSprites, 10);
                        }
                        else if(!onWater && currentAnimation != RIGHT) {
                                setAnimation(RIGHT, rightSprites, 10);
                        }
                }
                if(up) {
                        if(onWater && currentAnimation != UPBOAT) {
                                setAnimation(UPBOAT, upBoatSprites, 10);
                        }
                        else if(!onWater && currentAnimation != UP) {
                                setAnimation(UP, upSprites, 10);
                        }
                }

                super.update();

        }

        public void draw(Graphics2D g) {
                super.draw(g);
        }
```

```
        public void update() {

                ticks++;


                boolean current = onWater;
                if(tileMap.getIndex(ydest / tileSize, xdest / tileSize) == 4) {
                        onWater = true;
                }
                else {
                        onWater = false;
                }
                if(!current && onWater) {
                        JukeBox.play("splash");
                }


                if(down) {
                        if(onWater && currentAnimation != DOWNBOAT) {
                                setAnimation(DOWNBOAT, downBoatSprites, 10);
                        }
                        else if(!onWater && currentAnimation != DOWN) {
                                setAnimation(DOWN, downSprites, 10);
                        }
                }
                if(left) {
                        if(onWater && currentAnimation != LEFTBOAT) {
                                setAnimation(LEFTBOAT, leftBoatSprites, 10);
                        }
                        else if(!onWater && currentAnimation != LEFT) {
                                setAnimation(LEFT, leftSprites, 10);
```

Sparkle

```
8
9   public class Sparkle extends Entity {
10
11          private boolean remove;
12
13          public Sparkle(TileMap tm) {
14              super(tm);
15              animation.setFrames(Content.SPARKLE[0]);
16              animation.setDelay(5);
17              width = height = 16;
18          }
19
20          public boolean shouldRemove() {
21              return remove;
22          }
23
24          public void update() {
25              animation.update();
26              if(animation.hasPlayedOnce()) remove = true;
27          }
28
29          public void draw(Graphics2D g) {
30              super.draw(g);
31          }
32
33  }
34
```

Hud

```
17
18  public class Hud {
19
21      private int yoffset;
26
28      private BufferedImage bar;
29      private BufferedImage diamond;
30      private BufferedImage boat;
31      private BufferedImage axe;
33
34      private Player player;
36
37      private int numDiamonds;
39
40      private Font font;
41      private Color textColor;
43
44      public Hud(Player p, ArrayList<Diamond> d) {
46
47          player = p;
48          numDiamonds = d.size();
49          yoffset = GamePanel.HEIGHT;
51
52          bar = Content.BAR[0][0];
53          diamond = Content.DIAMOND[0][0];
54          boat = Content.ITEMS[0][0];
55          axe = Content.ITEMS[0][1];
57
58          font = new Font("Arial", Font.PLAIN, 10);
59          textColor = new Color(47, 64, 126);
61
62      }
64
65      public void draw(Graphics2D g) {
67
68          g.drawImage(bar, 0, yoffset, null);
70
```

```
49
50      public void draw(Graphics2D g) {
51
52
53          g.drawImage(bar, 0, yoffset, null);
54
55
56          g.setColor(textColor);
57          g.fillRect(8, yoffset + 6, (int)(28.0 * player.numDiamonds() / numDiamonds), 4);
58
59
60          g.setColor(textColor);
61          g.setFont(font);
62          String s = player.numDiamonds() + "/" + numDiamonds;
63          Content.drawString(g, s, 40, yoffset + 3);
64          if(player.numDiamonds() >= 10) g.drawImage(diamond, 80, yoffset, null);
65          else g.drawImage(diamond, 72, yoffset, null);
66
67
68          if(player.hasBoat()) g.drawImage(boat, 100, yoffset, null);
69          if(player.hasAxe()) g.drawImage(axe, 112, yoffset, null);
70
71
72          int minutes = (int) (player.getTicks() / 1800);
73          int seconds = (int) ((player.getTicks() / 30) % 60);
74          if(minutes < 10) {
75              if(seconds < 10) Content.drawString(g, "0" + minutes + ":0" + seconds, 85, 3);
76              else Content.drawString(g, "0" + minutes + ":" + seconds, 85, 3);
77          }
78          else {
79              if(seconds < 10) Content.drawString(g, minutes + ":0" + seconds, 85, 3);
80              else Content.drawString(g, minutes + ":" + seconds, 85, 3);
81          }
82
83
84
85      }
86
```

Tile

```
6   import java.awt.image.BufferedImage;
7
8   public class Tile {
9
10      private BufferedImage image;
11      private int type;
12
13
14      public static final int NORMAL = 0;
15      public static final int BLOCKED = 1;
16
17      public Tile(BufferedImage image, int type) {
18          this.image = image;
19          this.type = type;
20      }
21
22      public BufferedImage getImage() { return image; }
23      public int getType() { return type; }
24
25  }
26
```

Tile Map

```
public class TileMap {

        // position
        private int x;
        private int y;
        private int xdest;
        private int ydest;
        private int speed;
        private boolean moving;

        // bounds
        private int xmin;
        private int ymin;
        private int xmax;
        private int ymax;

        // map
        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 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;
                        xmin = -width;
                        xmax = 0;
                        ymin = GamePanel.HEIGHT - height;
                        ymin = -height;
                        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 void loadTiles(String s) {

                try {

                        tileset = ImageIO.read(
                                getClass().getResourceAsStream(s)
                        );
                        numTilesAcross = tileset.getWidth() / tileSize;
                        tiles = new Tile[2][numTilesAcross];

                        BufferedImage subimage;
                        for(int col = 0; col < numTilesAcross; col++) {
                                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();
                }
```

```
                }
                catch(Exception e) {
                        e.printStackTrace();
                }

        }

        public int getTileSize() { return tileSize; }
        public int getx() { return x; }
        public int gety() { return y; }
        public int getWidth() { return width; }
        public int getHeight() { return height; }
        public int getNumRows() { return numRows; }
        public int getNumCols() { return numCols; }
        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 int getIndex(int row, int col) {
                return map[row][col];
        }
        public boolean isMoving() { return moving; }

        public void setTile(int row, int col, int index) {
                map[row][col] = index;
        }
        public void replace(int i1, int i2) {
                for(int row = 0; row < numRows; row++) {
                        for(int col = 0; col < numCols; col++) {
                                if(map[row][col] == i1) map[row][col] = i2;
                        }
                }
        }
```

```
                colOffset = -this.x / tileSize;
                rowOffset = -this.y / tileSize;

                if(x != xdest || y != ydest) moving = true;
                else moving = false;

        }

        public void draw(Graphics2D g) {

                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(),
                                        x + col * tileSize,
                                        y + row * tileSize,
                                        null
                                );

                        }

                }

        }
```

Source Codes:

https://www.youtube.com/watch?v=AA1XpWHhxw0

https://github.com/foreignguymike/legacyYTtutorials/tree/master/DiamondHunter

Github Link:

https://github.com/ReiHajime/CC3---1E---StardustCrusader