



STARDUST CRUSADEERS

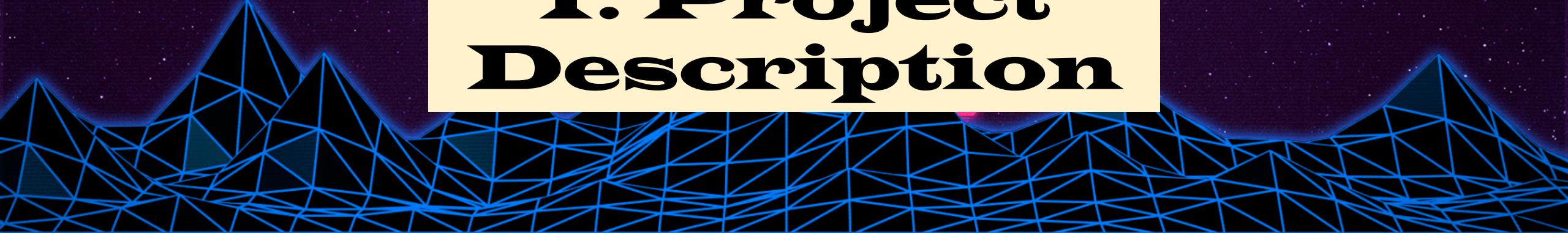


MEMBERS:

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

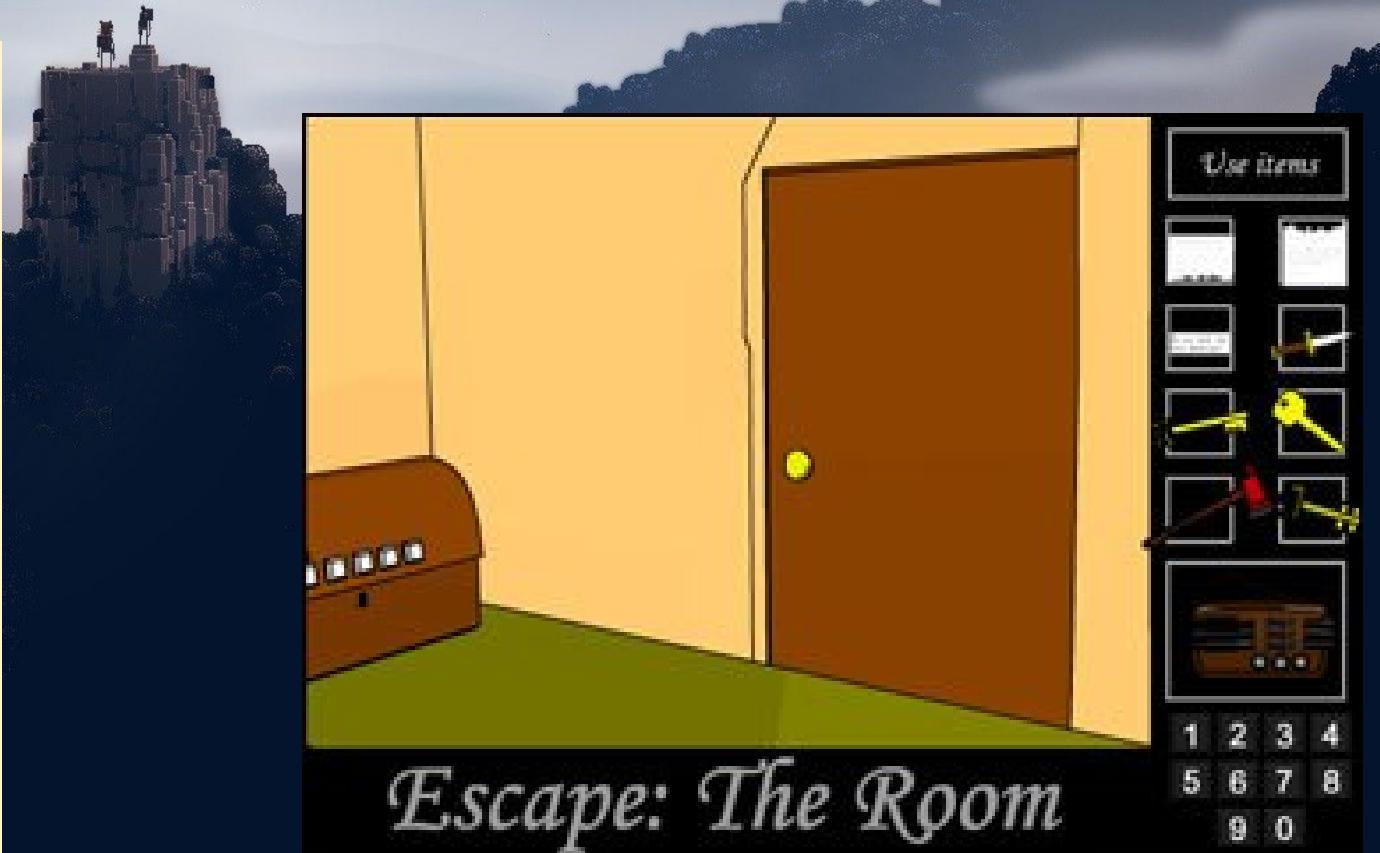
Based on:

- 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.



Escape: The Room



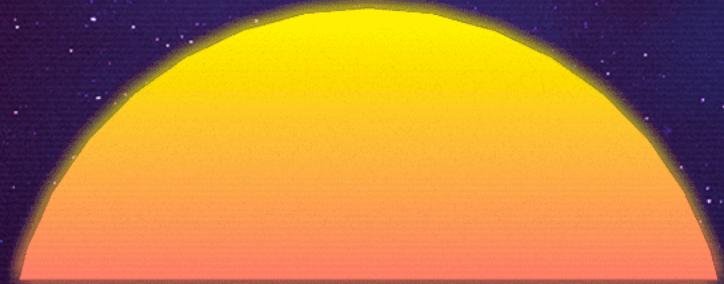
- RPG or Role-Playing Video Game genre began in the mid-1970s, which combined with the ruleset of fantasy wargames. A game in which the participants assume the roles of characters and collaboratively create stories.
- In our game, the player or the character will take the role as a stardust crusaders 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 Pokémons 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 finds 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.



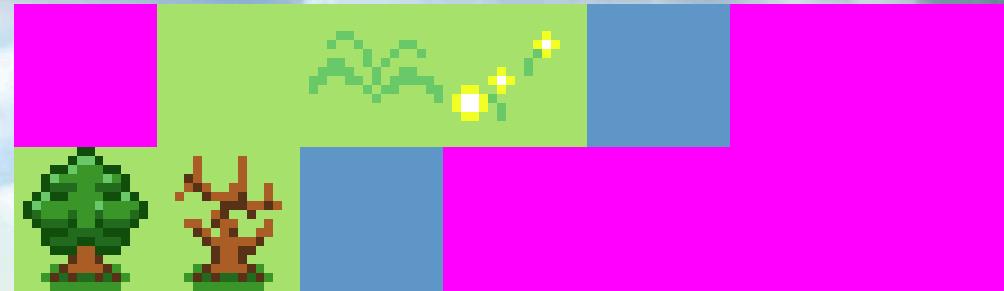
- The member of the knights templar, the adventurer, control this hero and collect all the magical orbs to grant 3 wishes you want.



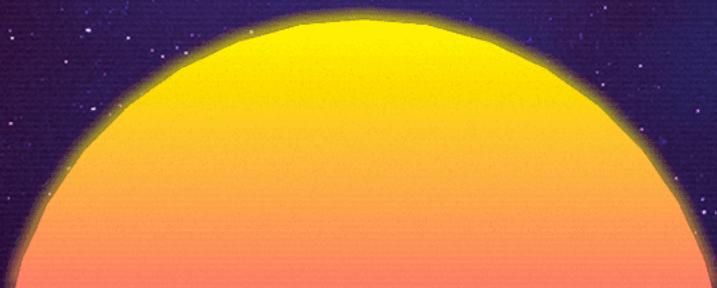
- The Magical Orb, Where it was created in the realm of the dragon world. Collect them all and win the game and your wishes can come true.



- 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.

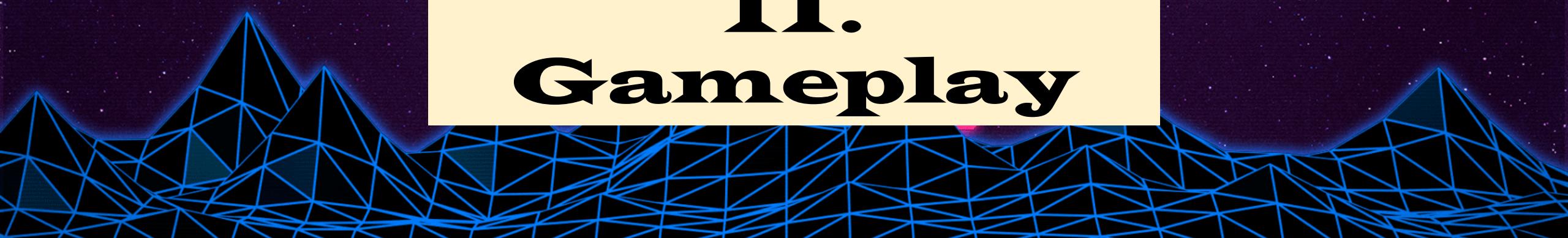


- The Tilemap, Fyke Isle is a small island in Velen, located in Lake Wyndamer. A green landscape, green and brown trees, and a blue lake water.

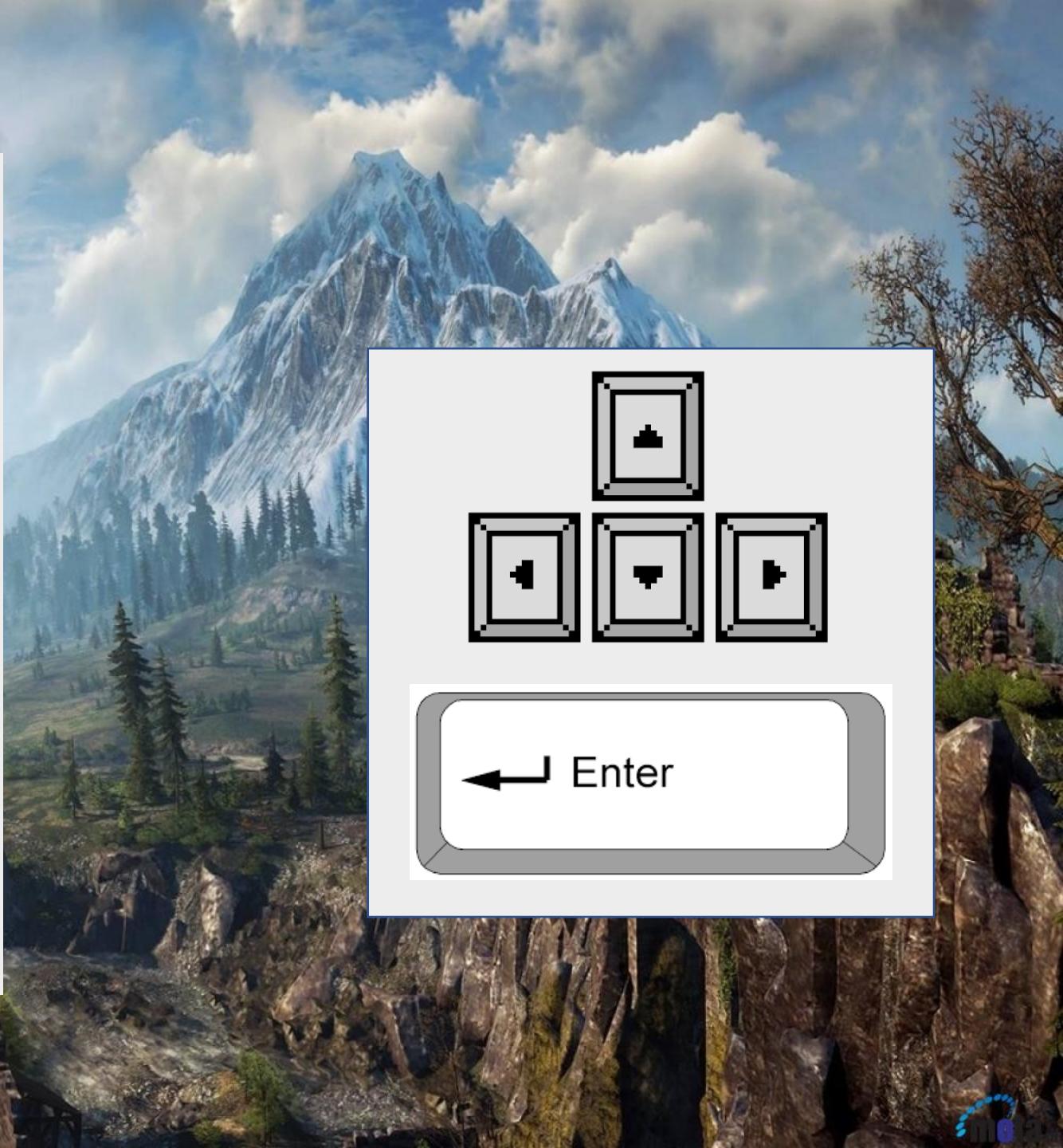


II.

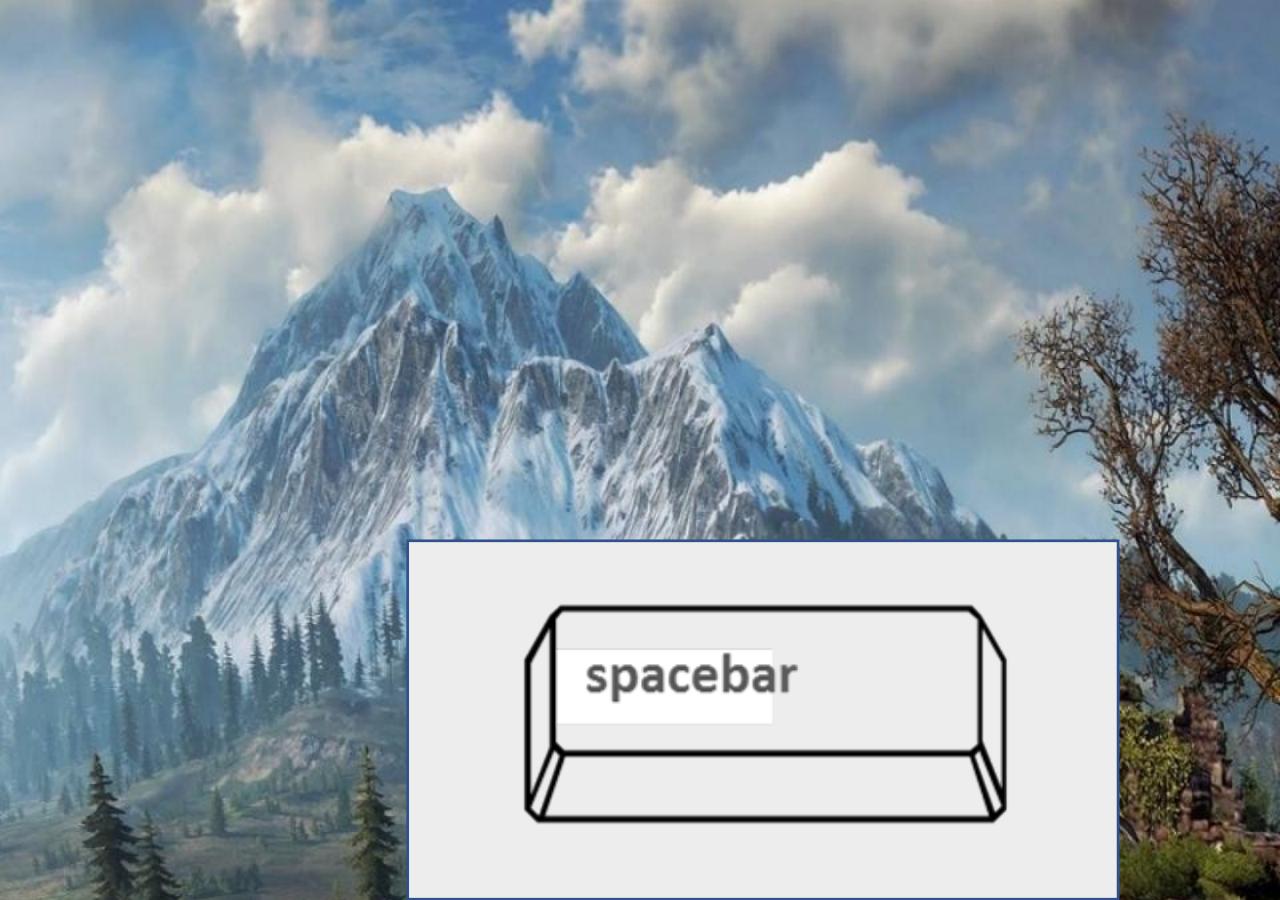
Gameplay



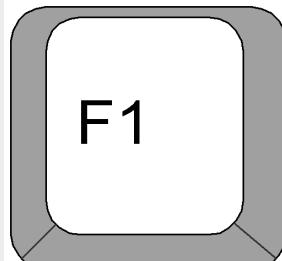
- Arrow Keys: Up, Down, Left, Right Keys: Makes the player move across the map. Also the controls for the menu screen.
- Press the Enter Key whether to start or exit the game at the menu screen.
- Start: will start the game and timer.
- Exit: will exit the game



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



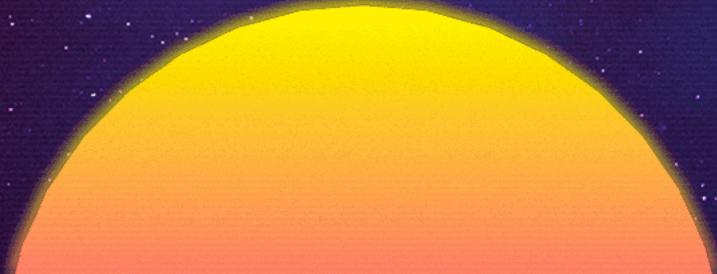
- ESC Key: can pause the game, and it will go to the pause screen. Where it shows the controls of the game which is the arrow keys, spacebar.
- Press the ESC Key again to unpause the game.
- F1 Key: Return To The Main Menu.



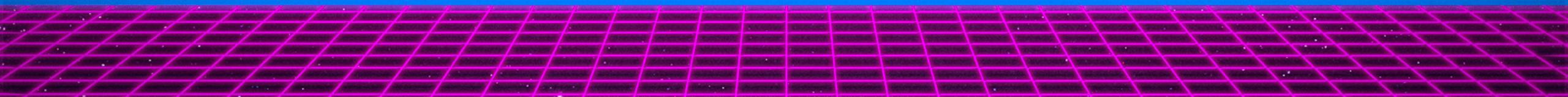
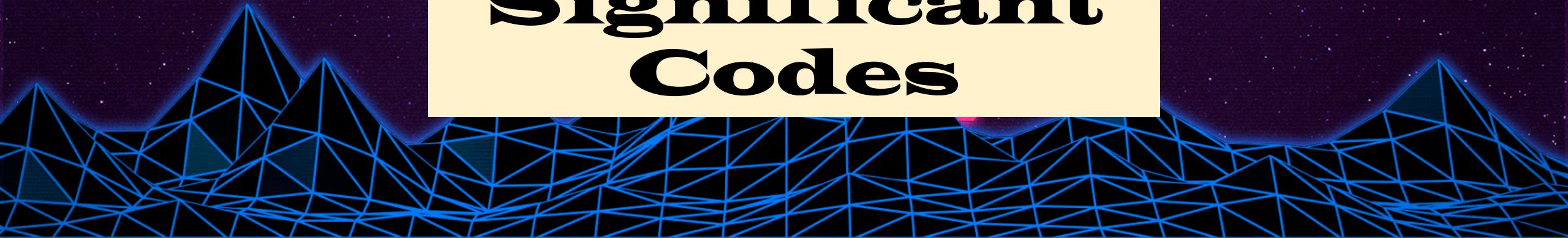
- How to Finish the Game:

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



Significant Codes



Game

```
3
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

```
19
20  @SuppressWarnings("serial")
21  public class GamePanel extends JPanel implements Runnable, KeyListener {
22
23      // dimensions
24      // HEIGHT is the playing area size
25      // HEIGHT2 includes the bottom window
26      public static final int WIDTH = 128;
27      public static final int HEIGHT = 128;
28      public static final int HEIGHT2 = HEIGHT + 16;
29      public static final int SCALE = 3;
30
31      // game loop stuff
32      private Thread thread;
33      private boolean running;
34      private final int FPS = 30;
35      private final int TARGET_TIME = 1000 / FPS;
36
37      // drawing stuff
38      private BufferedImage image;
39      private Graphics2D g;
40
41      // game state manager
42      private GameStateManager gsm;
43
44      // constructor
45      public GamePanel() {
46          setPreferredSize(new Dimension(WIDTH * SCALE, HEIGHT2 * SCALE));
47          setFocusable(true);
48          requestFocus();
49      }
50
```

```
84
85
86
87
88
89
90
91     private void init() {
92         running = true;
93         image = new BufferedImage(WIDTH, HEIGHT2, 1);
94         g = (Graphics2D) image.getGraphics();
95         gsm = new GameStateManager();
96     }
97
98
99     private void update() {
100        gsm.update();
101        Keys.update();
102    }
103
104    private void draw() {
105        gsm.draw(g);
106    }
107
108
109    private void drawToScreen() {
110        Graphics g2 = getGraphics();
111        g2.drawImage(image, 0, 0, WIDTH * SCALE, HEIGHT2 * SCALE, null);
112        g2.dispose();
113    }
114
115
116    public void keyTyped(KeyEvent key) {}
117    public void keyPressed(KeyEvent key) {
118        Keys.keySet(key.getKeyCode(), true);
119    }
120
121    public void keyReleased(KeyEvent key) {
122        Keys.keySet(key.getKeyCode(), false);
123    }
124
```

```
45
46     public void addNotify() {
47         super.addNotify();
48         if(thread == null) {
49             addKeyListener(this);
50             thread = new Thread(this);
51             thread.start();
52         }
53     }
54
55
56
57     public void run() {
58
59         init();
60
61         long start;
62         long elapsed;
63         long wait;
64
65
66         while(running) {
67
68             start = System.nanoTime();
69
70             update();
71             draw();
72             drawToScreen();
73
74             elapsed = System.nanoTime() - start;
75
76             wait = TARGET_TIME - elapsed / 1000000;
77             if(wait < 0) wait = TARGET_TIME;
78
79             try {
80                 Thread.sleep(wait);
81             } catch(Exception e) {
82                 e.printStackTrace();
83             }
84
85         }
86     }
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
```

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         }
38         catch(Exception e) {
39             e.printStackTrace();
40             System.out.println("Error loading graphics.");
41             System.exit(0);
42         }
43     }
44     return null;
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 }
```

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     public void setState(int i) {  
46         previousState = currentState;  
47         unloadState(previousState);  
48         currentState = i;  
49         if(i == INTRO){  
50             gameStates[i] = new IntroState(this);  
51             gameStates[i].init();  
52         }  
53         else if(i == MENU) {  
54     }
```

```
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

```
10 import javax.sound.sampled.AudioInputStream;
11 import javax.sound.sampled.AudioSystem;
12 import javax.sound.sampled.Clip;
13 import javax.sound.sampled.FloatControl;
14
15 public class JukeBox {
16
17     private static HashMap<String, Clip> clips;
18     private static int gap;
19
20
21     public static void init() {
22         clips = new HashMap<String, Clip>();
23         gap = 0;
24     }
25
26
27     public static void load(String s, String n) {
28         if(clips.get(n) != null) return;
29         Clip clip;
30         try {
31             InputStream in = JukeBox.class.getResourceAsStream(s);
32             InputStream bin = new BufferedInputStream(in);
33             AudioInputStream ais =
34                 AudioSystem.getAudioInputStream(bin);
35             AudioFormat baseFormat = ais.getFormat();
36             AudioFormat decodeFormat =
37                 new AudioFormat(
38                     AudioFormat.Encoding.PCM_SIGNED,
39                     baseFormat.getSampleRate(),
40                     16,
41                     baseFormat.getChannels(),
42                     baseFormat.getChannels() * 2,
43                     baseFormat.getSampleRate(),
44                     false
45                 );
46             AudioInputStream dais = AudioSystem.getAudioInputStream(
47                 clip = AudioSystem.getClip();
48                 clip.open(dais);
```

```
49
50     }
51     catch(Exception e) {
52         e.printStackTrace();
53     }
54
55     public static void play(String s) {
56         play(s, gap);
57     }
58
59     public static void play(String s, int i) {
60         Clip c = clips.get(s);
61         if(c == null) return;
62         if(c.isRunning()) c.stop();
63         c.setFramePosition(i);
64         while(!c.isRunning()) c.start();
65     }
66
67     public static void stop(String s) {
68         if(clips.get(s) == null) return;
69         if(clips.get(s).isRunning()) clips.get(s).stop();
70     }
71
72     public static void resume(String s) {
73         if(clips.get(s).isRunning()) return;
74         clips.get(s).start();
75     }
76
77     public static void resumeLoop(String s) {
78         Clip c = clips.get(s);
79         if(c == null) return;
80         c.loop(Clip.LOOP_CONTINUOUSLY);
81     }
82
83     public static void loop(String s) {
84         loop(s, gap, gap, clips.get(s).getFrameLength() - 1);
85
86
87     public static void loop(String s, int frame, int start, int end) {
88         Clip c = clips.get(s);
89         if(c == null) return;
90         if(c.isRunning()) c.stop();
91         c.setLoopPoints(start, end);
92         c setFramePosition(frame);
93         c.loop(Clip.LOOP_CONTINUOUSLY);
94     }
95
96     public static void setPosition(String s, int frame) {
97         clips.get(s).setFramePosition(frame);
98     }
99
100    public static int getFrames(String s) { return clips.get(s).getFrameLength(); }
101    public static int getPosition(String s) { return clips.get(s).getFramePosition(); }
102
103    public static void close(String s) {
104        stop(s);
105        clips.get(s).close();
106    }
107
108    public static void setVolume(String s, float f) {
109        Clip c = clips.get(s);
110        if(c == null) return;
111        FloatControl vol = (FloatControl) c.getControl(FloatControl.Type.MASTER_GAIN);
112        vol.setValue(f);
113    }
114
115    public static boolean isPlaying(String s) {
116        Clip c = clips.get(s);
117        if(c == null) return false;
118        return c.isRunning();
119    }
120
121
122
123
124
125
126
127
128
```

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) {  
32          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() {  
43          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
```

```
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() {  
43          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

```
11 import java.awt.Color;
12 import java.awt.Graphics2D;
13
14 import com.neet.DiamondHunter.Main.GamePanel;
15 import com.neet.DiamondHunter.Manager.Content;
16 import com.neet.DiamondHunter.Manager.Data;
17
18 import com.neet.DiamondHunter.Manager.GameStateManager;
19 import com.neet.DiamondHunter.Manager.JukeBox;
20 import com.neet.DiamondHunter.Manager.Keys;
21
22 public class GameOverState extends GameState {
23
24     private Color color;
25
26     private int rank;
27     private long ticks;
28
29     public GameOverState(GameStateManager gsm) {
30         super(gsm);
31     }
32
33     public void init() {
34         color = new Color(164, 198, 222);
35         ticks = Data.getTime();
36         if(ticks < 3600) rank = 1;
37         else if(ticks < 5400) rank = 2;
38         else if(ticks < 7200) rank = 3;
39         else rank = 4;
40     }
41
42     public void update() {}
43
44     public void draw(Graphics2D g) {
45
46         g.setColor(color);
47         g.fillRect(0, 0, GamePanel.WIDTH, GamePanel.HEIGHT2);
48
49         Content.drawString(g, "finish time", 20, 20);
50
51         int minutes = (int) (ticks / 1800);
52         int seconds = (int) ((ticks / 30) % 60);
53         if(minutes < 10) {
54             if(seconds < 10) Content.drawString(g, "0" + minutes + ":"0" + seconds, 44, 32);
55             else Content.drawString(g, "0" + minutes + ":" + seconds, 44, 32);
56         }
57         else {
58             if(seconds < 10) Content.drawString(g, minutes + ":"0" + seconds, 44, 32);
59             else Content.drawString(g, minutes + ":" + seconds, 44, 22);
60         }
61     }
62 }
```

```
51
52         int minutes = (int) (ticks / 1800);
53         int seconds = (int) ((ticks / 30) % 60);
54         if(minutes < 10) {
55             if(seconds < 10) Content.drawString(g, "0" + minutes + ":"0" + seconds, 44, 32);
56             else Content.drawString(g, "0" + minutes + ":" + seconds, 44, 32);
57         }
58         else {
59             if(seconds < 10) Content.drawString(g, minutes + ":"0" + seconds, 44, 32);
60             else Content.drawString(g, minutes + ":" + seconds, 44, 22);
61         }
62
63         Content.drawString(g, "rank", 48, 45);
64         if(rank == 1) Content.drawString(g, "God of Speed", 20, 58);
65         else if(rank == 2) Content.drawString(g, "adventurer", 24, 58);
66         else if(rank == 3) Content.drawString(g, "apprentice", 32, 58);
67         else if(rank == 4) Content.drawString(g, "too slow", 8, 58);
68
69         Content.drawString(g, "Your Three", 15, 75);
70         Content.drawString(g, "Wishes May", 15, 95);
71         Content.drawString(g, "Be Granted", 15, 115);
72     }
73
74     public void handleInput() {
75         if(Keys.isPressed(Keys.ENTER)) {
76             gsm.setState(GameStateManager.MENU);
77             JukeBox.play("collect");
78         }
79     }
80
81 }
```

Game State

```
1  public abstract class GameState {  
2  
3  
4      protected GameStateManager gsm;  
5  
6      [-]  
7      public GameState(GameStateManager gsm) {  
8          this.gsm = gsm;  
9      }  
10  
11      public abstract void init();  
12      public abstract void update();  
13      public abstract void draw(Graphics2D g);  
14      public abstract void handleInput();  
15  
16      }  
17  
18 }
```

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          } catch(Exception e) {
35              e.printStackTrace();
36          }
37      }
38
39      public void update() {
40          handleInput();
41          ticks++;
42          if(ticks < FADE_IN) {
43              alpha = (int) (255 - 255 * (1.0 * ticks / FADE_IN));
44          }
45      }
46
47      public void draw(Graphics2D g) {
48          g.setColor(Color.WHITE);
49          g.fillRect(0, 0, GamePanel.WIDTH, GamePanel.HEIGHT2);
50          g.drawImage(logo, 0, 0, GamePanel.WIDTH, GamePanel.HEIGHT2, null);
51          g.setColor(new Color(0, 0, 0, alpha));
52          g.fillRect(0, 0, GamePanel.WIDTH, GamePanel.HEIGHT2);
53      }
54
55      public void handleInput() {
56          if(Keys.isPressed(Keys.ENTER)) {
57              gsm.setState(GameStateManager.MENU);
58          }
59      }
60  }
```

```
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
62
63
64
65
66
67
68
69
70
```

```
if(ticks < FADE_IN) {
    alpha = (int) (255 - 255 * (1.0 * ticks / FADE_IN));
    if(alpha < 0) alpha = 0;
}
if(ticks > FADE_IN + LENGTH) {
    alpha = (int) (255 * (1.0 * ticks - FADE_IN - LENGTH) / FADE_OUT);
    if(alpha > 255) alpha = 255;
}
if(ticks > FADE_IN + LENGTH + FADE_OUT) {
    gsm.setState(GameStateManager.MENU);
}

public void draw(Graphics2D g) {
    g.setColor(Color.WHITE);
    g.fillRect(0, 0, GamePanel.WIDTH, GamePanel.HEIGHT2);
    g.drawImage(logo, 0, 0, GamePanel.WIDTH, GamePanel.HEIGHT2, null);
    g.setColor(new Color(0, 0, 0, alpha));
    g.fillRect(0, 0, GamePanel.WIDTH, GamePanel.HEIGHT2);

}

public void handleInput() {
    if(Keys.isPressed(Keys.ENTER)) {
        gsm.setState(GameStateManager.MENU);
    }
}
```

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          g.drawImage(bg, 0, 0, null);  
41  
42          Content.drawString(g, options[0], 44, 90);  
43          Content.drawString(g, options[1], 48, 100);  
44      }  
45  }
```

```
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
42     public void handleInput() {
43         if(Keys.isPressed(Keys.ESCAPE)) {
44             gsm.setPaused(false);
45             JukeBox.resumeLoop("music1");
46         }
47         if(Keys.isPressed(Keys.F1)) {
48             gsm.setPaused(false);
49             gsm.setState(GameStateManager.MENU);
50         }
51     }
52 }
```

Play State

```
20
21
22
23
24     public class PlayState extends GameState {
25
26
27         private Player player;
28
29         private TileMap tileMap;
30
31         private ArrayList<Diamond> diamonds;
32
33         private ArrayList<Item> items;
34
35         private ArrayList<Sparkle> sparkles;
36
37
38         private int xsector;
39         private int ysector;
40         private int sectorSize;
41
42         private Hud hud;
43
44
45         private boolean blockInput;
46         private boolean eventStart;
47         private boolean eventFinish;
48         private int eventTick;
49
50
51         player.setTilePosition(17, 17);
52         player.setTotalDiamonds(diamonds.size());
53
54
55         sectorSize = GamePanel.WIDTH;
56         xsector = player.gettx() / sectorSize;
57         ysector = player.getty() / sectorSize;
58         tileMap.setPositionImmediately(-xsector * sectorSize, -ysector * sectorSize);
59
60
61         hud = new Hud(player, diamonds);
62
63         JukeBox.load("/Music/bgmusic.mp3", "music1");
64         JukeBox.setVolume("music1", -10);
65         JukeBox.loop("music1", 1000, 1000, JukeBox.getFrames("music1") - 1000);
66         JukeBox.load("/Music/finish.mp3", "finish");
67         JukeBox.setVolume("finish", -10);
68
69
70         JukeBox.load("/SFX/collect.wav", "collect");
71         JukeBox.load("/SFX/mapmove.wav", "mapmove");
72         JukeBox.load("/SFX/tilechange.wav", "tilechange");
73         JukeBox.load("/SFX/splash.wav", "splash");
74
75
76         boxes = new ArrayList<Rectangle>();
77         eventStart = true;
78         eventStart();
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111 }
```

```
170
171
172     private void populateItems() {
173
174         Item item;
175
176         item = new Item(tileMap);
177         item.setType(Item.AXE);
178         item.setTitlePosition(26, 37);
179         items.add(item);
180
181         item = new Item(tileMap);
182         item.setType(Item.BOAT);
183         item.setTitlePosition(12, 4);
184         items.add(item);
185
186     }
187
188     public void update() {
189
190         handleInput();
191
192         if(eventStart) eventStart();
193         if(eventFinish) eventFinish();
194
195         if(player.numDiamonds() == player.getTotalDiamonds()) {
196             eventFinish = blockInput = true;
197         }
198
199
200     }
201
202
203
204
205         for(int i = 0; i < sparkles.size(); i++) {
206             Sparkle s = sparkles.get(i);
207             s.update();
208             if(s.shouldRemove()) {
209                 sparkles.remove(i);
210                 i--;
211             }
212
213         }
214
215         for(int i = 0; i < items.size(); i++) {
216             Item item = items.get(i);
217             if(player.intersects(item)) {
218                 items.remove(i);
219                 i--;
220                 item.collected(player);
221                 JukeBox.play("collect");
222                 Sparkle s = new Sparkle(tileMap);
223                 s.setPosition(item.gettx(), item.getty());
224                 sparkles.add(s);
225
226         }
227
228     }
229
230     public void draw(Graphics2D g) {
231
232         tileMap.draw(g);
233
234         player.draw(g);
235
236     }
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
759
760
761
762
763
764
765
766
767
768
769
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
787
788
789
789
790
791
792
793
794
795
796
797
797
798
799
799
800
801
802
803
804
805
806
807
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
827
828
829
829
830
831
832
833
833
834
835
835
836
837
837
838
839
839
840
841
841
842
843
843
844
845
845
846
847
847
848
849
849
850
851
851
852
853
853
854
855
855
856
857
857
858
859
859
860
861
861
862
863
863
864
865
865
866
867
867
868
869
869
870
871
871
872
873
873
874
875
875
876
877
877
878
879
879
880
881
881
882
883
883
884
885
885
886
887
887
888
889
889
890
891
891
892
893
893
894
895
895
896
897
897
898
899
899
900
901
901
902
903
903
904
905
905
906
907
907
908
909
909
910
911
911
912
913
913
914
915
915
916
917
917
918
919
919
920
921
921
922
923
923
924
925
925
926
927
927
928
929
929
930
931
931
932
933
933
934
935
935
936
937
937
938
939
939
940
941
941
942
943
943
944
945
945
946
947
947
948
949
949
950
951
951
952
953
953
954
955
955
956
957
957
958
959
959
960
961
961
962
963
963
964
965
965
966
967
967
968
969
969
970
971
971
972
973
973
974
975
975
976
977
977
978
979
979
980
981
981
982
983
983
984
985
985
986
987
987
988
989
989
990
991
991
992
993
993
994
995
995
996
997
997
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543
1544
1544
1545
1545
1546
1546
1547
1547
1548
1548
1549
1549
1550
1550
1551
1551
1552
1552
1553
1553
1554
1554
1555
1555
1556
1556
1557
1557
1558
1558
1559
1559
1560
1560
1561
1561
1562
1562
1563
1563
1564
1564
1565
1565
1566
1566
1567
1567
1568
1568
1569
1569
1570
1570
1571
1571
1572
1572
1573
1573
1574
1574
1575
1575
1576
1576
1577
1577
1578
1578
1579
1579
1580
1580
1581
1581
1582
1582
1583
1583
1584
1584
1585
1585
1586
1586
1587
1587
1588
1588
1589
1589
1590
1590
1591
1591
1592
1592
1593
1593
1594
1594
1595
1595
1596
1596
1597
1597
1598
1598
1599
1599
1600
1600
1601
1601
1602
1602
1603
1603
1604
1604
1605
1605
1606
1606
1607
1607
1608
1608
1609
1609
1610
1610
1611
1611
1612
1612
1613
1613
1614
1614
1615
1615
1616
1616
1617
1617
1618
1618
1619
1619
1620
1620
1621
1621
1622
1622
1623
1623
1624
1624
1625
1625
1626
1626
1627
1627
1628
1628
1629
1629
1630
1630
1631
1631
1632
1632
1633
1633
1634
1634
1635
1635
1636
1636
1637
1637
1638
1638
1639
1639
1640
1640
1641
1641
1642
1642
16
```

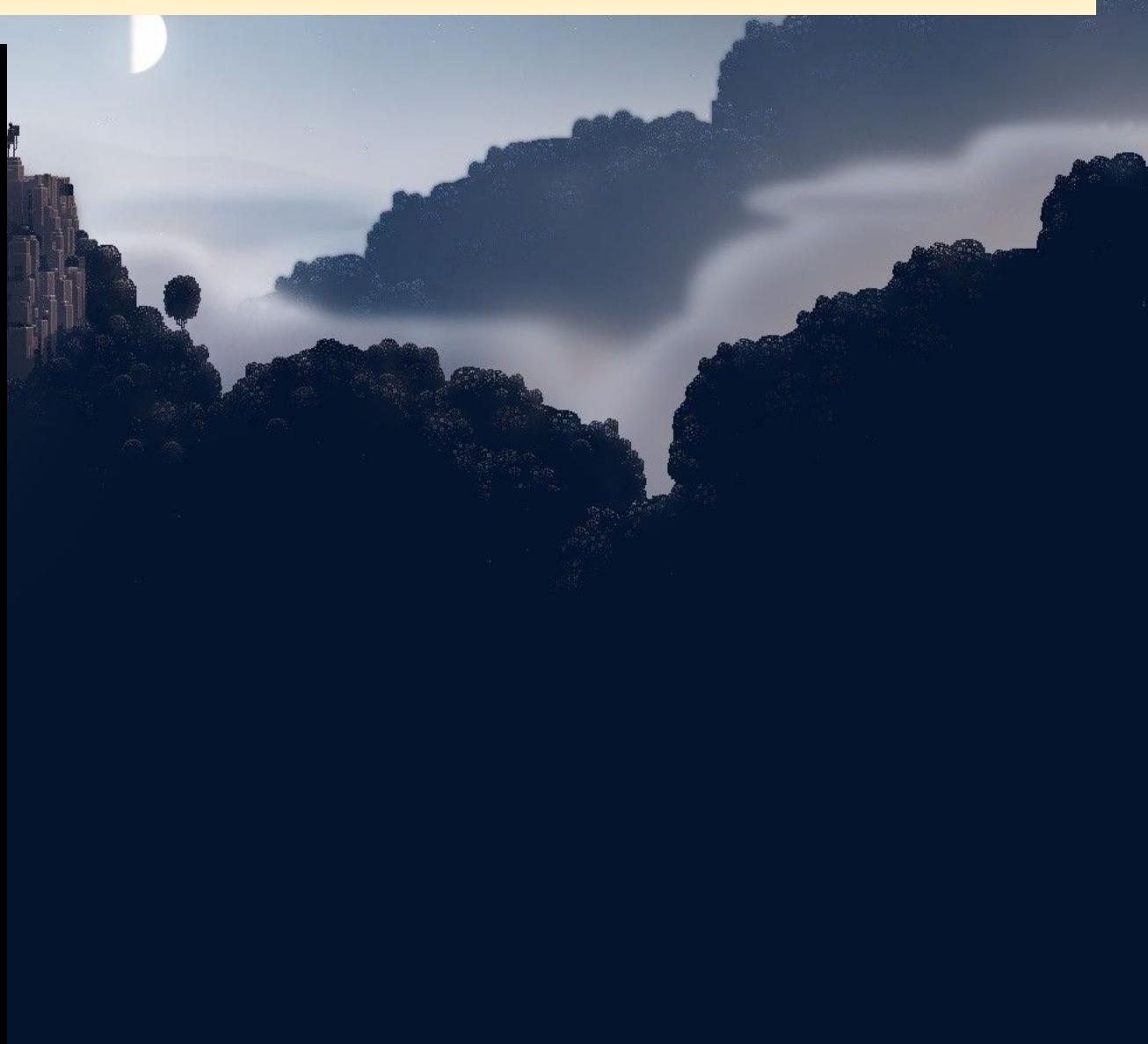
Animation

```
9  public class Animation {  
10  
11      private BufferedImage[] frames;  
12      private int currentFrame;  
13      private int numFrames;  
14  
15      private int count;  
16      private int delay;  
17  
18      private int timesPlayed;  
19  
20      public Animation() {  
21          timesPlayed = 0;  
22      }  
23  
24      public void setFrames(BufferedImage[] frames) {  
25          this.frames = frames;  
26          currentFrame = 0;  
27          count = 0;  
28          timesPlayed = 0;  
29          delay = 2;  
30          numFrames = frames.length;  
31      }  
32  
33      public void setDelay(int i) { delay = i; }  
34      public void setFrame(int i) { currentFrame = i; }  
35      public void setNumFrames(int i) { numFrames = i; }  
36  
37      public void update() {  
38  
39          if(delay == -1) return;
```

```
33      }  
34      }  
35      }  
36      }  
37      }  
38  
39      public void setDelay(int i) { delay = i; }  
40      public void setFrame(int i) { currentFrame = i; }  
41      public void setNumFrames(int i) { numFrames = i; }  
42  
43      public void update() {  
44  
45          if(count == delay) {  
46              currentFrame++;  
47              count = 0;  
48          }  
49          if(currentFrame == numFrames) {  
50              currentFrame = 0;  
51              timesPlayed++;  
52          }  
53  
54      }  
55  
56      public int getFrame() { return currentFrame; }  
57      public int getCount() { return count; }  
58      public BufferedImage getImage() { return frames[currentFrame]; }  
59      public boolean hasPlayedOnce() { return timesPlayed > 0; }  
60      public boolean hasPlayed(int i) { return timesPlayed == i; }  
61  }
```

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  }
```



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, cwidht, 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;
```

```
46     protected Animation animation;
47     protected int currentAnimation;
48
49     public Entity(TileMap tm) {
50         tileMap = tm;
51         tileSize = tileMap.getTileSize();
52         animation = new Animation();
53     }
54
55     public int getx() { return x; }
56     public int gety() { return y; }
57     public int getRow() { return rowTile; }
58     public int getCol() { return colTile; }
59
60     public void setPosition(int i1, int i2) {
61         x = i1;
62         y = i2;
63         xdest = x;
64         ydest = y;
65     }
66     public void setMapPosition() {
67         xmap = tileMap.getx();
68         ymap = tileMap.gety();
69     }
70     public void setTilePosition(int i1, int i2) {
71         y = i1 * tileSize + tileSize / 2;
72         x = i2 * tileSize + tileSize / 2;
73         xdest = x;
74         ydest = y;
75     }
76
77     public void setLeft() {
78
79     }
80 }
```

```
152
153     public void getNextPosition() {
154
155         if(left && x > xdest) x -= moveSpeed;
156         else left = false;
157         if(left && x < xdest) x = xdest;
158
159         if(right && x < xdest) x += moveSpeed;
160         else right = false;
161         if(right && x > xdest) x = xdest;
162
163         if(up && y > ydest) y -= moveSpeed;
164         else up = false;
165         if(up && y < ydest) y = ydest;
166
167         if(down && y < ydest) y += moveSpeed;
168         else down = false;
169         if(down && y > ydest) y = ydest;
170
171     }
172
173     public void update() {
174
175
176         if(moving) getNextPosition();
177
178
179         if(x == xdest && y == ydest) {
180             left = right = up = down = moving = false;
181             rowTile = y / tileSize;
182             colTile = x / tileSize;
183         }
184     }
185 }
```

```
174
175
176     if(moving) getNextPosition();
177
178     if(x == xdest && y == ydest) {
179         left = right = up = down = moving = false;
180         rowTile = y / tileSize;
181         colTile = x / tileSize;
182     }
183
184
185     animation.update();
186
187 }
188
189
190
191     public void draw(Graphics2D g) {
192         setMapPosition();
193         g.drawImage(
194             animation.getImage(),
195             x + xmap - width / 2,
196             y + ymap - height / 2,
197             null
198         );
199     }
200
201 }
```

Item

```
22     cwidth = cheight = 12;
23
24
25     public void setType(int i) {
26         type = i;
27         if(type == BOAT) {
28             sprite = Content.ITEMS[1][0];
29         }
30         else if(type == AXE) {
31             sprite = Content.ITEMS[1][1];
32         }
33     }
34
35     public void collected(Player p) {
36         if(type == BOAT) {
37             p.gotBoat();
38         }
39         if(type == AXE) {
40             p.gotAxe();
41         }
42     }
43
44     public void draw(Graphics2D g) {
45         setMapPosition();
46         g.drawImage(sprite, x + xmap - width / 2, y + ymap - height / 2, null);
47     }
48
49 }
50 }
```

```
10
11     public class Item extends Entity{
12
13         private BufferedImage sprite;
14         private int type;
15         public static final int BOAT = 0;
16         public static final int AXE = 1;
17
18         public Item(TileMap tm) {
19             super(tm);
20             type = -1;
21             width = height = 16;
22             cwidth = cheight = 12;
23         }
24
25         public void setType(int i) {
26             type = i;
27             if(type == BOAT) {
28                 sprite = Content.ITEMS[1][0];
29             }
30             else if(type == AXE) {
31                 sprite = Content.ITEMS[1][1];
32             }
33         }
34
35         public void collected(Player p) {
36             if(type == BOAT) {
37                 p.gotBoat();
38             }
39             if(type == AXE) {
40                 p.gotAxe();
41             }
42         }
43
44     }
45 }
```

Player

```
10 public class Player extends Entity {
11
12     private BufferedImage[] downSprites;
13     private BufferedImage[] upSprites;
14     private BufferedImage[] rightSprites;
15     private BufferedImage[] leftSprites;
16     private BufferedImage[] downBoatSprites;
17     private BufferedImage[] leftBoatSprites;
18     private BufferedImage[] upBoatSprites;
19
20     private final int DOWN = 0;
21     private final int LEFT = 1;
22     private final int RIGHT = 2;
23     private final int UP = 3;
24     private final int DOWNBOAT = 4;
25     private final int LEFTBOAT = 5;
26     private final int RIGHTBOAT = 6;
27     private final int UBOAT = 7;
28
29     private int numDiamonds;
30     private int totalDiamonds;
31     private boolean hasBoat;
32     private boolean hasAxe;
33     private boolean onWater;
34
35     private long ticks;
36
37     public Player(TileMap tm) {
38
39     }
40 }
```

```
67     private void setAnimation(int i, BufferedImage[] bi, int d) {
68         currentAnimation = i;
69         animation.setFrames(bi);
70         animation.setDelay(d);
71     }
72
73     public void collectedDiamond() { numDiamonds++; }
74     public int numDiamonds() { return numDiamonds; }
75     public int getTotalDiamonds() { return totalDiamonds; }
76     public void setTotalDiamonds(int i) { totalDiamonds = i; }
77
78     public void gotBoat() { hasBoat = true; tileMap.replace(22, 4); }
79     public void gotAxe() { hasAxe = true; }
80     public boolean hasBoat() { return hasBoat; }
81     public boolean hasAxe() { return hasAxe; }
82
83
84     public long getTicks() { return ticks; }
85
86     public void setDown() {
87         super.setDown();
88     }
89     public void setLeft() {
90         super.setLeft();
91     }
92     public void setRight() {
93         super.setRight();
94     }
95     public void setUp() {
96         super.setUp();
97     }
98
99 }
```

```
148
149     if(onWater && currentAnimation != LEFTBOAT) {
150         setAnimation(LEFTBOAT, leftBoatSprites, 10);
151     }
152     else if(!onWater && currentAnimation != LEFT) {
153         setAnimation(LEFT, leftSprites, 10);
154     }
155
156     if(right) {
157         if(onWater && currentAnimation != RIGHTBOAT) {
158             setAnimation(RIGHTBOAT, rightBoatSprites, 10);
159         }
160         else if(!onWater && currentAnimation != RIGHT) {
161             setAnimation(RIGHT, rightSprites, 10);
162         }
163     }
164     if(up) {
165         if(onWater && currentAnimation != UPBOAT) {
166             setAnimation(UPBOAT, upBoatSprites, 10);
167         }
168         else if(!onWater && currentAnimation != UP) {
169             setAnimation(UP, upSprites, 10);
170         }
171     }
172
173     super.update();
174
175
176     public void draw(Graphics2D g) {
177         super.draw(g);
178     }
179 }
```

```
40
41     public Player(TileMap tm) {
42
43         super(tm);
44
45         width = 16;
46         height = 16;
47         cwidth = 12;
48         cheight = 12;
49
50         moveSpeed = 2;
51
52         numDiamonds = 0;
53
54         downSprites = Content.PLAYER[0];
55         leftSprites = Content.PLAYER[1];
56         rightSprites = Content.PLAYER[2];
57         upSprites = Content.PLAYER[3];
58         downBoatSprites = Content.PLAYER[4];
59         leftBoatSprites = Content.PLAYER[5];
60         rightBoatSprites = Content.PLAYER[6];
61         upBoatSprites = Content.PLAYER[7];
62
63         animation.setFrames(downSprites);
64         animation.setDelay(10);
65
66     }
67     private void setAnimation(int i, BufferedImage[] bi, int d) {
68         currentAnimation = i;
69         animation.setFrames(bi);
70         animation.setDelay(d);
71     }
72
73
74
75
76
77
78
79
80
81
82
83
84
85
86
87
88
89
90
91
92
93
94
95
96
97
98
99
100
101
102
103
104
105
106
107
108
109
110
111
112
113
114
115
116
117
118
119
120
121
122
123
124
125
126
127
128
129
130
131
132
133
134
135
136
137
138
139
140
141
142
143
144
145
146
147
148
149
150
151
152
153
154
155
156
157
158
159
160
161
162
163
164
165
166
167
168
169
170
171
172
173
174
175
176
177
178
179
180
181
182
183
184
185
186
187
188
189
190
191
192
193
194
195
196
197
198
199
200
201
202
203
204
205
206
207
208
209
210
211
212
213
214
215
216
217
218
219
220
221
222
223
224
225
226
227
228
229
230
231
232
233
234
235
236
237
238
239
240
241
242
243
244
245
246
247
248
249
250
251
252
253
254
255
256
257
258
259
260
261
262
263
264
265
266
267
268
269
270
271
272
273
274
275
276
277
278
279
280
281
282
283
284
285
286
287
288
289
290
291
292
293
294
295
296
297
298
299
300
301
302
303
304
305
306
307
308
309
310
311
312
313
314
315
316
317
318
319
320
321
322
323
324
325
326
327
328
329
330
331
332
333
334
335
336
337
338
339
340
341
342
343
344
345
346
347
348
349
350
351
352
353
354
355
356
357
358
359
360
361
362
363
364
365
366
367
368
369
370
371
372
373
374
375
376
377
378
379
380
381
382
383
384
385
386
387
388
389
390
391
392
393
394
395
396
397
398
399
400
401
402
403
404
405
406
407
408
409
410
411
412
413
414
415
416
417
418
419
420
421
422
423
424
425
426
427
428
429
430
431
432
433
434
435
436
437
438
439
440
441
442
443
444
445
446
447
448
449
450
451
452
453
454
455
456
457
458
459
460
461
462
463
464
465
466
467
468
469
470
471
472
473
474
475
476
477
478
479
480
481
482
483
484
485
486
487
488
489
490
491
492
493
494
495
496
497
498
499
500
501
502
503
504
505
506
507
508
509
510
511
512
513
514
515
516
517
518
519
520
521
522
523
524
525
526
527
528
529
530
531
532
533
534
535
536
537
538
539
540
541
542
543
544
545
546
547
548
549
550
551
552
553
554
555
556
557
558
559
560
561
562
563
564
565
566
567
568
569
570
571
572
573
574
575
576
577
578
579
580
581
582
583
584
585
586
587
588
589
590
591
592
593
594
595
596
597
598
599
600
601
602
603
604
605
606
607
608
609
610
611
612
613
614
615
616
617
618
619
620
621
622
623
624
625
626
627
628
629
630
631
632
633
634
635
636
637
638
639
640
641
642
643
644
645
646
647
648
649
650
651
652
653
654
655
656
657
658
659
660
661
662
663
664
665
666
667
668
669
670
671
672
673
674
675
676
677
678
679
680
681
682
683
684
685
686
687
688
689
690
691
692
693
694
695
696
697
698
699
700
701
702
703
704
705
706
707
708
709
710
711
712
713
714
715
716
717
718
719
720
721
722
723
724
725
726
727
728
729
730
731
732
733
734
735
736
737
738
739
740
741
742
743
744
745
746
747
748
749
750
751
752
753
754
755
756
757
758
759
760
761
762
763
764
765
766
767
768
769
770
771
772
773
774
775
776
777
778
779
779
780
781
782
783
784
785
786
787
788
789
789
790
791
792
793
794
795
796
797
798
799
800
801
802
803
804
805
806
807
808
809
809
810
811
812
813
814
815
816
817
818
819
819
820
821
822
823
824
825
826
827
828
829
829
830
831
832
833
834
835
836
837
837
838
839
839
840
841
842
843
844
844
845
846
846
847
847
848
848
849
849
850
850
851
851
852
852
853
853
854
854
855
855
856
856
857
857
858
858
859
859
860
860
861
861
862
862
863
863
864
864
865
865
866
866
867
867
868
868
869
869
870
870
871
871
872
872
873
873
874
874
875
875
876
876
877
877
878
878
879
879
880
880
881
881
882
882
883
883
884
884
885
885
886
886
887
887
888
888
889
889
890
890
891
891
892
892
893
893
894
894
895
895
896
896
897
897
898
898
899
899
900
900
901
901
902
902
903
903
904
904
905
905
906
906
907
907
908
908
909
909
910
910
911
911
912
912
913
913
914
914
915
915
916
916
917
917
918
918
919
919
920
920
921
921
922
922
923
923
924
924
925
925
926
926
927
927
928
928
929
929
930
930
931
931
932
932
933
933
934
934
935
935
936
936
937
937
938
938
939
939
940
940
941
941
942
942
943
943
944
944
945
945
946
946
947
947
948
948
949
949
950
950
951
951
952
952
953
953
954
954
955
955
956
956
957
957
958
958
959
959
960
960
961
961
962
962
963
963
964
964
965
965
966
966
967
967
968
968
969
969
970
970
971
971
972
972
973
973
974
974
975
975
976
976
977
977
978
978
979
979
980
980
981
981
982
982
983
983
984
984
985
985
986
986
987
987
988
988
989
989
990
990
991
991
992
992
993
993
994
994
995
995
996
996
997
997
998
998
999
999
1000
1000
1001
1001
1002
1002
1003
1003
1004
1004
1005
1005
1006
1006
1007
1007
1008
1008
1009
1009
1010
1010
1011
1011
1012
1012
1013
1013
1014
1014
1015
1015
1016
1016
1017
1017
1018
1018
1019
1019
1020
1020
1021
1021
1022
1022
1023
1023
1024
1024
1025
1025
1026
1026
1027
1027
1028
1028
1029
1029
1030
1030
1031
1031
1032
1032
1033
1033
1034
1034
1035
1035
1036
1036
1037
1037
1038
1038
1039
1039
1040
1040
1041
1041
1042
1042
1043
1043
1044
1044
1045
1045
1046
1046
1047
1047
1048
1048
1049
1049
1050
1050
1051
1051
1052
1052
1053
1053
1054
1054
1055
1055
1056
1056
1057
1057
1058
1058
1059
1059
1060
1060
1061
1061
1062
1062
1063
1063
1064
1064
1065
1065
1066
1066
1067
1067
1068
1068
1069
1069
1070
1070
1071
1071
1072
1072
1073
1073
1074
1074
1075
1075
1076
1076
1077
1077
1078
1078
1079
1079
1080
1080
1081
1081
1082
1082
1083
1083
1084
1084
1085
1085
1086
1086
1087
1087
1088
1088
1089
1089
1090
1090
1091
1091
1092
1092
1093
1093
1094
1094
1095
1095
1096
1096
1097
1097
1098
1098
1099
1099
1100
1100
1101
1101
1102
1102
1103
1103
1104
1104
1105
1105
1106
1106
1107
1107
1108
1108
1109
1109
1110
1110
1111
1111
1112
1112
1113
1113
1114
1114
1115
1115
1116
1116
1117
1117
1118
1118
1119
1119
1120
1120
1121
1121
1122
1122
1123
1123
1124
1124
1125
1125
1126
1126
1127
1127
1128
1128
1129
1129
1130
1130
1131
1131
1132
1132
1133
1133
1134
1134
1135
1135
1136
1136
1137
1137
1138
1138
1139
1139
1140
1140
1141
1141
1142
1142
1143
1143
1144
1144
1145
1145
1146
1146
1147
1147
1148
1148
1149
1149
1150
1150
1151
1151
1152
1152
1153
1153
1154
1154
1155
1155
1156
1156
1157
1157
1158
1158
1159
1159
1160
1160
1161
1161
1162
1162
1163
1163
1164
1164
1165
1165
1166
1166
1167
1167
1168
1168
1169
1169
1170
1170
1171
1171
1172
1172
1173
1173
1174
1174
1175
1175
1176
1176
1177
1177
1178
1178
1179
1179
1180
1180
1181
1181
1182
1182
1183
1183
1184
1184
1185
1185
1186
1186
1187
1187
1188
1188
1189
1189
1190
1190
1191
1191
1192
1192
1193
1193
1194
1194
1195
1195
1196
1196
1197
1197
1198
1198
1199
1199
1200
1200
1201
1201
1202
1202
1203
1203
1204
1204
1205
1205
1206
1206
1207
1207
1208
1208
1209
1209
1210
1210
1211
1211
1212
1212
1213
1213
1214
1214
1215
1215
1216
1216
1217
1217
1218
1218
1219
1219
1220
1220
1221
1221
1222
1222
1223
1223
1224
1224
1225
1225
1226
1226
1227
1227
1228
1228
1229
1229
1230
1230
1231
1231
1232
1232
1233
1233
1234
1234
1235
1235
1236
1236
1237
1237
1238
1238
1239
1239
1240
1240
1241
1241
1242
1242
1243
1243
1244
1244
1245
1245
1246
1246
1247
1247
1248
1248
1249
1249
1250
1250
1251
1251
1252
1252
1253
1253
1254
1254
1255
1255
1256
1256
1257
1257
1258
1258
1259
1259
1260
1260
1261
1261
1262
1262
1263
1263
1264
1264
1265
1265
1266
1266
1267
1267
1268
1268
1269
1269
1270
1270
1271
1271
1272
1272
1273
1273
1274
1274
1275
1275
1276
1276
1277
1277
1278
1278
1279
1279
1280
1280
1281
1281
1282
1282
1283
1283
1284
1284
1285
1285
1286
1286
1287
1287
1288
1288
1289
1289
1290
1290
1291
1291
1292
1292
1293
1293
1294
1294
1295
1295
1296
1296
1297
1297
1298
1298
1299
1299
1300
1300
1301
1301
1302
1302
1303
1303
1304
1304
1305
1305
1306
1306
1307
1307
1308
1308
1309
1309
1310
1310
1311
1311
1312
1312
1313
1313
1314
1314
1315
1315
1316
1316
1317
1317
1318
1318
1319
1319
1320
1320
1321
1321
1322
1322
1323
1323
1324
1324
1325
1325
1326
1326
1327
1327
1328
1328
1329
1329
1330
1330
1331
1331
1332
1332
1333
1333
1334
1334
1335
1335
1336
1336
1337
1337
1338
1338
1339
1339
1340
1340
1341
1341
1342
1342
1343
1343
1344
1344
1345
1345
1346
1346
1347
1347
1348
1348
1349
1349
1350
1350
1351
1351
1352
1352
1353
1353
1354
1354
1355
1355
1356
1356
1357
1357
1358
1358
1359
1359
1360
1360
1361
1361
1362
1362
1363
1363
1364
1364
1365
1365
1366
1366
1367
1367
1368
1368
1369
1369
1370
1370
1371
1371
1372
1372
1373
1373
1374
1374
1375
1375
1376
1376
1377
1377
1378
1378
1379
1379
1380
1380
1381
1381
1382
1382
1383
1383
1384
1384
1385
1385
1386
1386
1387
1387
1388
1388
1389
1389
1390
1390
1391
1391
1392
1392
1393
1393
1394
1394
1395
1395
1396
1396
1397
1397
1398
1398
1399
1399
1400
1400
1401
1401
1402
1402
1403
1403
1404
1404
1405
1405
1406
1406
1407
1407
1408
1408
1409
1409
1410
1410
1411
1411
1412
1412
1413
1413
1414
1414
1415
1415
1416
1416
1417
1417
1418
1418
1419
1419
1420
1420
1421
1421
1422
1422
1423
1423
1424
1424
1425
1425
1426
1426
1427
1427
1428
1428
1429
1429
1430
1430
1431
1431
1432
1432
1433
1433
1434
1434
1435
1435
1436
1436
1437
1437
1438
1438
1439
1439
1440
1440
1441
1441
1442
1442
1443
1443
1444
1444
1445
1445
1446
1446
1447
1447
1448
1448
1449
1449
1450
1450
1451
1451
1452
1452
1453
1453
1454
1454
1455
1455
1456
1456
1457
1457
1458
1458
1459
1459
1460
1460
1461
1461
1462
1462
1463
1463
1464
1464
1465
1465
1466
1466
1467
1467
1468
1468
1469
1469
1470
1470
1471
1471
1472
1472
1473
1473
1474
1474
1475
1475
1476
1476
1477
1477
1478
1478
1479
1479
1480
1480
1481
1481
1482
1482
1483
1483
1484
1484
1485
1485
1486
1486
1487
1487
1488
1488
1489
1489
1490
1490
1491
1491
1492
1492
1493
1493
1494
1494
1495
1495
1496
1496
1497
1497
1498
1498
1499
1499
1500
1500
1501
1501
1502
1502
1503
1503
1504
1504
1505
1505
1506
1506
1507
1507
1508
1508
1509
1509
1510
1510
1511
1511
1512
1512
1513
1513
1514
1514
1515
1515
1516
1516
1517
1517
1518
1518
1519
1519
1520
1520
1521
1521
1522
1522
1523
1523
1524
1524
1525
1525
1526
1526
1527
1527
1528
1528
1529
1529
1530
1530
1531
1531
1532
1532
1533
1533
1534
1534
1535
1535
1536
1536
1537
1537
1538
1538
1539
1539
1540
1540
1541
1541
1542
1542
1543
1543

```

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  }
```

Hud

```
17  public class Hud {
18
19
20      private int yoffset;
21
22      private BufferedImage bar;
23      private BufferedImage diamond;
24      private BufferedImage boat;
25      private BufferedImage axe;
26
27      private Player player;
28
29      private int numDiamonds;
30
31      private Font font;
32      private Color textColor;
33
34      public Hud(Player p, ArrayList<Diamond> d) {
35
36          player = p;
37          numDiamonds = d.size();
38          yoffset = GamePanel.HEIGHT;
39
40          bar = Content.BAR[0][0];
41          diamond = Content.DIAMOND[0][0];
42          boat = Content.ITEMS[0][0];
43          axe = Content.ITEMS[0][1];
44
45          font = new Font("Arial", Font.PLAIN, 10);
46          textColor = new Color(47, 64, 126);
47
48      }
49
50      public void draw(Graphics2D g) {
51
52
53          g.drawImage(bar, 0, yoffset, null);
54
55      }
56
57  }
```

```
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
11
12
13
14      private BufferedImage image;
15      private int type;
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

```
16 public class TileMap {
17     // position
18     private int x;
19     private int y;
20     private int xdest;
21     private int ydest;
22     private int speed;
23     private boolean moving;
24
25     // bounds
26     private int xmin;
27     private int ymin;
28     private int xmax;
29     private int ymax;
30
31     // map
32     private int[][] map;
33     private int numRows;
34     private int numCols;
35     private int width;
36     private int height;
37
38     // tileset
39     private BufferedImage tileset;
40     private int numTilesAcross;
41     private Tile[][] tiles;
42
43     // drawing
44     private int rowOffset;
45     private int colOffset;
46     private int numRowsToDelete;
47     private int numColsToDelete;
```

```
94     public void loadMap(String s) {
95
96         try {
97
98             InputStream in = getClass().getResourceAsStream(s);
99             BufferedReader br = new BufferedReader(
100                 new InputStreamReader(in));
101
102             numCols = Integer.parseInt(br.readLine());
103             numRows = Integer.parseInt(br.readLine());
104             map = new int[numRows][numCols];
105             width = numCols * tileSize;
106             height = numRows * tileSize;
107
108             xmin = GamePanel.WIDTH - width;
109             xmin = -width;
110             xmax = 0;
111             ymin = GamePanel.HEIGHT - height;
112             ymin = -height;
113             ymax = 0;
114
115             String delims = "\\\\s+";
116             for(int row = 0; row < numRows; row++) {
117                 String line = br.readLine();
118                 String[] tokens = line.split(delims);
119                 for(int col = 0; col < numCols; col++) {
120                     map[row][col] = Integer.parseInt(tokens[col]);
121                 }
122             }
123
124         } catch(Exception e) {
125             e.printStackTrace();
126         }
127     }
128 }
```

```
36     }
37
38     public void loadTiles(String s) {
39
40         try {
41
42             tileset = ImageIO.read(
43                 getClass().getResourceAsStream(s));
44
45             numTilesAcross = tileset.getWidth() / tileSize;
46             tiles = new Tile[2][numTilesAcross];
47
48             BufferedImage subimage;
49             for(int col = 0; col < numTilesAcross; col++) {
50                 subimage = tileset.getSubimage(
51                     col * tileSize,
52                     0,
53                     tileSize,
54                     tileSize
55                 );
56                 tiles[0][col] = new Tile(subimage, Tile.NORMAL);
57                 subimage = tileset.getSubimage(
58                     col * tileSize,
59                     tileSize,
60                     tileSize,
61                     tileSize
62                 );
63                 tiles[1][col] = new Tile(subimage, Tile.BLOCKED);
64             }
65         } catch(Exception e) {
66             e.printStackTrace();
67         }
68     }
69
70 }
```

```
124     }
125     catch(Exception e) {
126         e.printStackTrace();
127     }
128
129 }
130
131     public int getTileSize() { return tileSize; }
132     public int getx() { return x; }
133     public int gety() { return y; }
134     public int getWidth() { return width; }
135     public int getHeight() { return height; }
136     public int getNumRows() { return numRows; }
137     public int getNumCols() { return numCols; }
138     public int getGetType(int row, int col) {
139         int rc = map[row][col];
140         int r = rc / numTilesAcross;
141         int c = rc % numTilesAcross;
142         return tiles[r][c].getType();
143     }
144     public int getIndex(int row, int col) {
145         return map[row][col];
146     }
147     public boolean isMoving() { return moving; }
148
149     public void setTile(int row, int col, int index) {
150         map[row][col] = index;
151     }
152
153     public void replace(int il, int i0) {
154         for(int row = 0; row < numRows; row++) {
155             for(int col = 0; col < numCols; col++) {
156                 if(map[row][col] == il) map[row][col] = i0;
157             }
158         }
159     }
160
161     colOffset = -this.x / tileSize;
162     rowOffset = -this.y / tileSize;
163
164     if(x != xdest || y != ydest) moving = true;
165     else moving = false;
166
167 }
168
169     public void draw(Graphics2D g) {
170
171         for(int row = rowOffset; row < rowOffset + numRowsToDraw; row++) {
172
173             if(row >= numRows) break;
174
175             for(int col = colOffset; col < colOffset + numColsToDelete; col++) {
176
177                 if(col >= numCols) break;
178                 if(map[row][col] == 0) continue;
179
180                 int rc = map[row][col];
181                 int r = rc / numTilesAcross;
182                 int c = rc % numTilesAcross;
183
184                 g.drawImage(
185                     tiles[r][c].getImage(),
186                     x + col * tileSize,
187                     y + row * tileSize,
188                     null
189                 );
190             }
191         }
192     }
193
194 }
```

Source Codes:

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

[https://github.com/foreignguymike/legacyYTtutorials/
tree/master/DiamondHunter](https://github.com/foreignguymike/legacyYTtutorials/tree/master/DiamondHunter)

Github Link:

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