File - C:\Users\anon\Desktop\hmonics\core\src\view\GdxGame.java

```java
package view;

import com.badlogic.gdx.Game;
import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.graphics.OrthographicCamera;
import com.badlogic.gdx.graphics.g2d.SpriteBatch;
import com.badlogic.gdx.utils.viewport.ExtendViewport;
import model.DataManager;
import view.screens.TeamLogoSplashScreen;

/**
 * Used by libGDX to draw everything on the currently set screen.
 */
public class GdxGame extends Game {
    public static int virtualWidth = 1920;
    public static int virtualHeight = 1080;
    public static boolean isResolution43 = false;

    public OrthographicCamera camera;
    public ExtendViewport viewport;
    public SpriteBatch batch;

    public void create() {
        if (((float) Gdx.app.getGraphics().getWidth() / (float) Gdx.app.getGraphics()
.getHeight()) == (4f / 3f)) {
            isResolution43 = true;
            virtualHeight = 1440;
        }
        camera = new OrthographicCamera();
        camera.viewportHeight = virtualHeight;
        camera.viewportWidth = virtualWidth;
        camera.setToOrtho(false, virtualWidth, virtualHeight);
        viewport = new ExtendViewport(virtualWidth, virtualHeight, virtualWidth, 1440
, camera);
        batch = new SpriteBatch();

        AssetManager.init();
        DataManager.populate();

        setScreen(new TeamLogoSplashScreen(GdxGame.this));
    }

    public void render() {
        super.render();
    } //important!

    public void dispose() {
        batch.dispose();
    }
}

```

File - C:\Users\anon\Desktop\hmonics\core\src\view\AssetManager.java

```java
package view;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.audio.Music;
import com.badlogic.gdx.graphics.Color;
import com.badlogic.gdx.graphics.g2d.BitmapFont;
import com.badlogic.gdx.graphics.g2d.TextureAtlas;
import com.badlogic.gdx.graphics.g2d.TextureRegion;
import com.badlogic.gdx.graphics.g2d.freetype.FreeTypeFontGenerator;
import com.badlogic.gdx.scenes.scene2d.ui.Button.ButtonStyle;
import com.badlogic.gdx.scenes.scene2d.ui.ImageButton.ImageButtonStyle;
import com.badlogic.gdx.scenes.scene2d.ui.Label.LabelStyle;
import com.badlogic.gdx.scenes.scene2d.ui.Skin;
import com.badlogic.gdx.scenes.scene2d.ui.TextButton.TextButtonStyle;
import com.badlogic.gdx.scenes.scene2d.ui.TextField.TextFieldStyle;
import com.badlogic.gdx.scenes.scene2d.utils.Drawable;
import com.badlogic.gdx.scenes.scene2d.utils.TextureRegionDrawable;

/**
 * Stores all of the assets for use with libGDX.
 */
public class AssetManager {
    public static TextureAtlas atlas;
    public static Skin defaultSkin;
    public static ButtonStyle defaultStyle;
    public static Drawable backPlate;
    public static BitmapFont font64;
    public static BitmapFont font32;
    public static LabelStyle labelStyle64Clear;
    public static LabelStyle labelStyle64Solid;
    public static TextFieldStyle textFieldStyle64;
    public static TextButtonStyle textButtonStyle64Checked;
    public static TextButtonStyle textButtonStyle64;
    public static TextButtonStyle textButtonStyle32;
    public static ImageButtonStyle imageButtonStyle;
    public static ImageButtonStyle backButtonStyle;

    /**
     * Initialize all of the asset styles for libGDX.
     */
    public static void init() {
        atlas = new TextureAtlas(Gdx.files.internal("packed-images/pack.atlas"));
        defaultSkin = new Skin(Gdx.files.internal("skins/clean-crispy/clean-crispy-ui.json"));
        defaultStyle = new ButtonStyle();
        defaultStyle.up = AssetManager.defaultSkin.getDrawable("button-c");
        defaultStyle.down = AssetManager.defaultSkin.getDrawable("button-pressed-over-c");
        defaultStyle.checked = AssetManager.defaultSkin.getDrawable("button-pressed-over-c");
        defaultStyle.over = AssetManager.defaultSkin.getDrawable("button-over-c");
        backPlate = AssetManager.defaultSkin.getDrawable("button-c-clear");
        // Fonts
```

```
51          FreeTypeFontGenerator generator = new FreeTypeFontGenerator(Gdx.files.
    internal("fonts/open-sans/OpenSans-Semibold.ttf"));
52          FreeTypeFontGenerator.FreeTypeFontParameter parameter = new
    FreeTypeFontGenerator.FreeTypeFontParameter();
53          parameter.hinting = FreeTypeFontGenerator.Hinting.Full;
54          parameter.color = Color.BLACK;
55          parameter.size = 64;
56          font64 = generator.generateFont(parameter);
57          parameter.size = 32;
58          font32 = generator.generateFont(parameter);
59          // Labels
60          labelStyle64Solid = new LabelStyle(font64, Color.BLACK);
61          labelStyle64Clear = new LabelStyle(font64, Color.BLACK);
62          labelStyle64Clear.background = view.AssetManager.backPlate;
63          // Text Fields
64          textFieldStyle64 = new TextFieldStyle(font64, Color.BLACK, defaultStyle.down
    , defaultStyle.down, defaultStyle.up);
65          // Text Buttons
66          textButtonStyle64Checked = new TextButtonStyle(defaultStyle.up, defaultStyle
    .down, defaultStyle.checked, font64);
67          textButtonStyle64 = new TextButtonStyle(defaultStyle.up, defaultStyle.down,
    defaultStyle.over, font64);
68          textButtonStyle32 = new TextButtonStyle(defaultStyle.up, defaultStyle.down,
    defaultStyle.over, font32);
69          //Image Buttons
70          imageButtonStyle = new ImageButtonStyle(defaultStyle);
71          backButtonStyle = new ImageButtonStyle(defaultStyle);
72          backButtonStyle.imageUp = new TextureRegionDrawable(getTextureRegion("
    BackButton"));
73      }
74
75      /**
76       * Retrieve a texture region from the texture atlas by the file name.
77       */
78      public static TextureRegion getTextureRegion(String fileName) {
79          return new TextureRegion(AssetManager.atlas.findRegion(fileName));
80      }
81
82      /**
83       * Retrieve an mp3 by the file name.
84       */
85      public static Music getMusic(String fileName) {
86          Music music = Gdx.audio.newMusic(Gdx.files.internal("sounds/" + fileName +
    ".mp3"));
87          music.setLooping(true);
88          music.setVolume(0.1f);
89          return music;
90      }
91  }
92
```

```java
package view.games;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.Screen;
import com.badlogic.gdx.audio.Music;
import com.badlogic.gdx.audio.Sound;
import com.badlogic.gdx.graphics.GL30;
import com.badlogic.gdx.math.Interpolation;
import com.badlogic.gdx.math.Vector2;
import com.badlogic.gdx.scenes.scene2d.*;
import com.badlogic.gdx.scenes.scene2d.actions.Actions;
import com.badlogic.gdx.scenes.scene2d.ui.*;
import com.badlogic.gdx.scenes.scene2d.utils.ChangeListener;
import com.badlogic.gdx.scenes.scene2d.utils.DragAndDrop;
import com.badlogic.gdx.scenes.scene2d.utils.TextureRegionDrawable;
import com.badlogic.gdx.utils.Align;
import model.Word;
import view.AssetManager;
import view.GdxGame;
import view.actors.Letter;
import view.screens.TeamLogoSplashScreen;
import viewmodel.ScreenManager;
import viewmodel.SpellingGameManager;

import java.util.ArrayList;
import java.util.Random;

public class SpellingGameScreen implements Screen {
    private GdxGame game;
    private Stage stage;
    private DragAndDrop dragAndDrop;
    private Random random;

    // Actors added to the screen are drawn in the order they were added. Actors drawn later are drawn on top of everything before.
    // These groups are used to add actors to the screen in the right order. All actors added to groups are drawn when the group is drawn.
    private Group backgroundGroup;
    private Group actorsGroup;
    private Group animationsGroup;

    private Table letterTable;
    private Table pictureTable;
    private Table spaceTable;
    private Container<Image> pictureContainer;
    private ArrayList<Container<Letter>> letterSpaces;
    public ImageButton backButton;
    public Label hintPopup;
    private TextButton skipButton;
    private TextButton hintButton;
    private SpellingGameManager spellingGameManager;
    private Music backgroundMusic;
    private Sound clickSound;
```

```
 52
 53      private int pictureSize = 400;
 54      private int letterSpaceWidth = 150;
 55      private int letterSpaceHeight = 195;
 56      private int letterSize = 140;
 57      private int buttonWidth = 300;
 58      private int buttonHeight = 150;
 59
 60      public SpellingGameScreen(GdxGame gdxGame) {
 61          this.game = gdxGame;
 62          stage = new Stage(gdxGame.viewport, gdxGame.batch);
 63          Gdx.input.setInputProcessor(stage);
 64          dragAndDrop = new DragAndDrop();
 65          dragAndDrop.setDragTime(0);
 66          dragAndDrop.setDragActorPosition(letterSize / 2, -letterSize / 2);
 67          random = new Random(System.currentTimeMillis());
 68
 69          setStage();
 70      }
 71
 72      /**
 73       * Screen size in virtual pixels: virtualWidth = 1920 virtualHeight = 1080;
 74       */
 75      private void setStage() {
 76          TeamLogoSplashScreen.getBackgroundMusic().stop();
 77          backgroundMusic = AssetManager.getMusic("GameMusic");
 78          backgroundMusic.setVolume(0.05f);
 79          backgroundMusic.play();
 80          clickSound = Gdx.audio.newSound(Gdx.files.internal("sounds/SFX/LetterClick.
    mp3"));
 81
 82          stage.addActor(backgroundGroup = new Group());
 83          stage.addActor(actorsGroup = new Group());
 84          stage.addActor(animationsGroup = new Group());
 85
 86          Image backgroundImage = new Image(view.AssetManager.getTextureRegion("
    background"));
 87          backgroundImage.setSize(GdxGame.virtualWidth, GdxGame.virtualHeight);
 88          backgroundGroup.addActor(backgroundImage);
 89
 90          Table mainTable = new Table();
 91          mainTable.setBounds(0, 0, GdxGame.virtualWidth, GdxGame.virtualHeight);
 92          actorsGroup.addActor(mainTable);
 93
 94          // Move picture table to fit the Hmong alphabet
 95          spaceTable = new Table();
 96          spaceTable.setBounds(mainTable.getWidth() / 2, 50, 0, letterSpaceHeight);
 97          mainTable.addActor(spaceTable);
 98          pictureTable = new Table();
 99          pictureTable.setBounds(mainTable.getWidth() / 2, 50 + letterSpaceHeight + 20
    , 0, pictureSize);
100          mainTable.addActor(pictureTable);
101          letterTable = new Table();
```

```
102         mainTable.add(letterTable).expand().top().padTop(20);
103 
104         pictureTable.add(pictureContainer = new Container<Image>().size(pictureSize)
    );
105         letterSpaces = new ArrayList<Container<Letter>>();
106 
107         // Back button
108         backButton = new ImageButton(AssetManager.backButtonStyle);
109         backButton.setBounds(50, 50, buttonHeight, buttonHeight);
110         backButton.addListener(new ChangeListener() {
111             @Override
112             public void changed(ChangeEvent event, Actor actor) {
113                 backgroundMusic.stop();
114                 TeamLogoSplashScreen.getBackgroundMusic().play();
115                 ScreenManager.setScreen(ScreenManager.getPreviousScreen());
116             }
117         });
118         backButton.setSize(buttonHeight, buttonHeight);
119         mainTable.addActor(backButton);
120 
121         // Skip button
122         skipButton = new TextButton("Skip", AssetManager.textButtonStyle64);
123         skipButton.setBounds(mainTable.getWidth() - buttonWidth - 50, 50,
    buttonWidth, buttonHeight);
124         skipButton.addListener(new ChangeListener() {
125             @Override
126             public void changed(ChangeEvent event, Actor actor) {
127                 SpellingGameScreen.this.spellingGameManager.changeToNextWord();
128             }
129         });
130         mainTable.addActor(skipButton);
131 
132         // Hint button
133         hintButton = new TextButton("Hint", AssetManager.textButtonStyle64);
134         hintButton.setBounds(mainTable.getWidth() - buttonWidth - 50, buttonHeight +
     50, buttonWidth, buttonHeight);
135         hintButton.addListener(new ChangeListener() {
136             @Override
137             public void changed(ChangeEvent event, Actor actor) {
138                 hintPopup.clearActions();
139                 hintPopup.addAction(Actions.sequence(
140                         Actions.fadeIn(2),
141                         Actions.delay(2),
142                         Actions.fadeOut(2)
143                 ));
144             }
145         });
146         mainTable.addActor(hintButton);
147 
148         hintPopup = new Label("", AssetManager.labelStyle64Clear);
149         hintPopup.setBounds(mainTable.getWidth() - buttonWidth - 50, buttonHeight *
    2 + 50, buttonWidth, buttonHeight);
150         hintPopup.getColor().a = 0;
```

```
151         hintPopup.setAlignment(Align.center);
152         mainTable.addActor(hintPopup);
153 
154         // Start complementary state machine last
155         spellingGameManager = new SpellingGameManager(this);
156     }
157 
158     public void setDisplayLanguage(ScreenManager.Language language) {
159         setAlphabet(language);
160     }
161 
162     /**
163      * Sets up game screen with indicated language and associated alphabet
164      */
165     private void setAlphabet(ScreenManager.Language language) {
166         int numRows = 3;
167         int numCol = 5;
168         int letterSelectSize = 89;
169         if (GdxGame.isResolution43) {
170             letterSelectSize += 20;
171             numRows += 2;
172             numCol -= 5;
173         }
174         letterTable.clearChildren();
175         switch (language) {
176             case ENGLISH:
177                 numRows = 2;
178                 letterSelectSize += 31;
179                 String[] alphabet = {
180                         "a", "b", "c", "d", "e", "f", "g", "h", "i", "j", "k", "l",
    "m",
181                         "n", "o", "p", "q", "r", "s", "t", "u", "v", "w", "x", "y",
    "z"};
182                 for (int i = 0; i < numRows; i++) { // row
183                     for (int j = 0; j < 13; j++) { // column
184                         Letter letter = new Letter(alphabet[(i * 13) + j],
    letterSelectSize);
185                         letterTable.add(new Container<Letter>(letter).size(
    letterSelectSize));
186                         setLetterAsDraggable(letter);
187                     }
188                     letterTable.row();
189                 }
190                 return;
191             case HMONG:
192                 numRows += 1;
193                 String[] consonants = {"c", "ch", "d", "dh", "dl", "f", "h", "hl", "
    hm", "hml", "hn", "hny",
194                         "k", "kh", "l", "m", "ml", "n", "nc", "nch", "ndl", "nk", "
    nkh", "np", "nph", "npl", "nplh", "nq",
195                         "nqh", "nr", "nrh", "nt", "nth", "nts", "ntsh", "ntx", "ntxh
    ", "ny", "p", "ph", "pl", "plh", "q",
196                         "qh", "r", "rh", "s", "t", "th", "ts", "tsh", "tx", "txh", "
```

```
196 v", "x", "xy", "y", "z"};
197                 String[] vowels = {"a", "aa", "ai", "au", "aw", "e", "ee", "i", "ia"
    , "o", "oo", "u", "ua", "w"};
198                 String[] tones = {"koJ", "muS", "kuV", "niaM", "neeG", "siaB", "zoO"
    , "toD"};
199
200                 Table consonantsTable = new Table();
201                 consonantsTable.setBackground(AssetManager.backPlate);
202                 letterTable.add(consonantsTable);
203                 Table vowelsTable = new Table();
204                 vowelsTable.setBackground(AssetManager.backPlate);
205                 letterTable.add(vowelsTable);
206                 Table tonesTable = new Table();
207                 tonesTable.setBackground(AssetManager.backPlate);
208                 letterTable.add(tonesTable);
209
210                 for (int i = 0; i < numRows; i++) { // row
211                     for (int j = 0; j < 10 + numCol; j++) { // column
212                         if ((i * (10 + numCol)) + j < consonants.length) { // leaves
     empty spaces
213                             Letter letter = new Letter(consonants[(i * (10 + numCol)
    ) + j], letterSelectSize - 10);
214                             consonantsTable.add(new Container<Letter>(letter).size(
    letterSelectSize));
215                             setLetterAsDraggable(letter);
216                         }
217                     }
218                     consonantsTable.row();
219                     for (int j = 0; j < 4; j++) { // column
220                         if ((i * 4) + j < vowels.length) { // leaves empty spaces
221                             Letter letter = new Letter(vowels[(i * 4) + j],
    letterSelectSize - 10);
222                             vowelsTable.add(new Container<Letter>(letter).size(
    letterSelectSize));
223                             setLetterAsDraggable(letter);
224                         }
225                     }
226                     vowelsTable.row();
227                     for (int j = 0; j < 2; j++) { // column
228                         if ((i * 2) + j < tones.length) { // leaves empty spaces
229                             Letter letter = new Letter(tones[(i * 2) + j],
    letterSelectSize - 10);
230                             letter.setIsTone();
231                             tonesTable.add(new Container<Letter>(letter).size(
    letterSelectSize));
232                             setLetterAsDraggable(letter);
233                         }
234                     }
235                     tonesTable.row();
236                 }
237                 return;
238         }
239     }
```

```
240
241     /**
242      * Creates a copy when letter is dragged from the alphabet. The copy does not
    create a copy when moved.
243      */
244     private void setLetterAsDraggable(Letter letter) {
245         dragAndDrop.addSource(new DragAndDrop.Source(letter) {
246             public DragAndDrop.Payload dragStart(InputEvent event, float x, float y,
     int pointer) {
247                 DragAndDrop.Payload payload = new DragAndDrop.Payload();
248                 Letter letterCopy = new Letter((Letter) getActor(), letterSize);
249                 payload.setDragActor(letterCopy);
250
251                 dragAndDrop.addSource(new DragAndDrop.Source(letterCopy) {
252                     @Override
253                     public DragAndDrop.Payload dragStart(InputEvent event, float x,
    float y, int pointer) {
254                         DragAndDrop.Payload payload = new DragAndDrop.Payload();
255                         payload.setDragActor(getActor());
256                         return payload;
257                     }
258                 });
259                 return payload;
260             }
261         });
262     }
263
264     public void setPictureAndSpaceLength(String pictureFileName, int spaceLength) {
265         pictureContainer.setActor(new Image(view.AssetManager.getTextureRegion(
    pictureFileName)));
266         setSpaces(spaceLength);
267     }
268
269     private void setSpaces(int spaceLength) {
270         spaceTable.clearChildren();
271         letterSpaces.clear();
272         for (int i = 0; i < spaceLength; i++) {
273             Container<Letter> letterContainer = new Container<Letter>();
274             letterContainer.setTouchable(Touchable.enabled);
275             letterContainer.setBackground(new TextureRegionDrawable(view.
    AssetManager.getTextureRegion("underline")));
276             spaceTable.add(letterContainer.size(letterSize, letterSize)).size(
    letterSpaceWidth, letterSpaceHeight);
277             letterSpaces.add(letterContainer);
278
279             dragAndDrop.addTarget(new DragAndDrop.Target(letterContainer) {
280                 @Override
281                 public boolean drag(DragAndDrop.Source source, DragAndDrop.Payload
    payload, float x, float y, int pointer) {
282                     return true;
283                 }
284
285                 @Override
```

```
286                 public void drop(DragAndDrop.Source source, DragAndDrop.Payload
    payload, float x, float y, int pointer) {
287                     for (Container<Letter> letterContainer : letterSpaces) {
288                         if (!letterContainer.hasChildren()) {
289                             letterContainer.setActor(null);
290                         }
291                     }
292                     Container<Letter> newParent = (Container<Letter>) getActor();
293                     newParent.setActor((Letter) payload.getDragActor());
294                     SpellingGameScreen.this.spellingGameManager.droppedLetter(
    payload.getDragActor());
295                 }
296             });
297         }
298     }
299
300     /**
301      * Gets the string formed by the letters dropped into the spaces
302      */
303     public String getWordInSpaces() {
304         String currentString = "";
305         for (Container<Letter> letterContainer : letterSpaces) {
306             if (letterContainer.hasChildren()) {
307                 currentString += letterContainer.getActor().getSpelling();
308             } else {
309                 currentString += " ";
310             }
311         }
312         return currentString;
313     }
314
315     /**
316      * Decides whether the spaces are filled with letter
317      */
318     public boolean spacesFull() {
319         for (Container<Letter> letterContainer : letterSpaces) {
320             if (letterContainer.hasChildren()) {
321                 continue;
322             } else {
323                 return false;
324             }
325         }
326         return true;
327     }
328
329     public void winConfetti(String fileName) {
330         for (Container<Letter> letterContainer : letterSpaces) {
331             if (letterContainer.hasChildren()) {
332                 confettiEffect(letterContainer.getActor(), fileName);
333             }
334         }
335     }
336
```

```
337     /**
338      * Confetti animation from the center of the subject actor.
339      */
340     public void confettiEffect(Actor subject, String fileName) {
341         int size = 100;
342         float duration = 1.5f;
343         int distance = 300;
344         Vector2 vector2 = subject.localToStageCoordinates(new Vector2(subject.getX()
    , subject.getY()));
345         for (int i = 0; i < 10; i++) {
346             Actor explosion = new Image(AssetManager.getTextureRegion(fileName));
347             explosion.setTouchable(Touchable.disabled);
348             animationsGroup.addActor(explosion);
349 
350             explosion.setBounds(vector2.x, vector2.y, size, size);
351             explosion.setOrigin(size / 2, size / 2);
352             explosion.addAction(Actions.moveTo(vector2.x + (random.nextBoolean() ?
    random.nextInt(distance) : -random.nextInt(distance)) + random.nextInt(distance),
    vector2.y + random.nextInt(distance),
353                     duration, Interpolation.smooth));
354             explosion.addAction(Actions.rotateBy(random.nextBoolean() ? random.
    nextInt(270) : -random.nextInt(270), duration));
355             explosion.addAction(Actions.fadeOut(duration));
356         }
357     }
358 
359     public void playLetter(String language, Letter letter) {
360         Sound sound = Gdx.audio.newSound(Gdx.files.internal("sounds/" + language + "
    Alphabet/" + letter.getName() + ".mp3"));
361         sound.play();
362     }
363 
364     public void playWord(String language, Word currentWord) {
365         Sound sound = Gdx.audio.newSound(Gdx.files.internal("sounds/" + language + "
    Words/" + currentWord.getWordId() + ".mp3"));
366         sound.play();
367     }
368 
369     public void playSFX(String sound) {
370         Sound sfx = Gdx.audio.newSound(Gdx.files.internal("sounds/SFX/" + sound + ".
    mp3"));
371         sfx.play();
372     }
373 
374     @Override
375     public void render(float delta) {
376         Gdx.gl.glClearColor(0, 0, 0, 1);
377         Gdx.gl.glClear(GL30.GL_COLOR_BUFFER_BIT);
378         stage.act(delta);
379         stage.draw();
380     }
381 
382     @Override
```

```
383     public void resize(int width, int height) {
384         stage.getViewport().update(width, height, false);
385     }
386 
387     @Override
388     public void show() {
389         Gdx.input.setInputProcessor(stage);
390     }
391 
392     @Override
393     public void hide() {
394     }
395 
396     @Override
397     public void pause() {
398     }
399 
400     @Override
401     public void resume() {
402     }
403 
404     @Override
405     public void dispose() {
406         stage.dispose();
407     }
408 }
409 
```

File - C:\Users\anon\Desktop\hmonics\core\src\view\actors\Letter.java

```java
package view.actors;

import com.badlogic.gdx.scenes.scene2d.ui.TextButton;
import view.AssetManager;

public class Letter extends TextButton {
    // Can have special spelling for tones.
    private String spelling;

    public Letter(String name, int size) {
        super(name, AssetManager.textButtonStyle64);
        if (size < 100) {
            setStyle(AssetManager.textButtonStyle32);
        }
        setName(name);
        spelling = name;
        setSize(size, size);
    }

    public Letter(Letter letter, int size) {
        this(letter.getName(), size);
        spelling = letter.spelling;
    }

    // If tone, trim actual spelling to last lowercase character.
    public void setIsTone() {
        spelling = getName().substring(getName().length() - 1).toLowerCase();
    }

    public String getSpelling() {
        return spelling;
    }
}

```

File - C:\Users\anon\Desktop\hmonics\core\src\view\screens\StartScreen.java

```
 1 package view.screens;
 2
 3 import com.badlogic.gdx.Gdx;
 4 import com.badlogic.gdx.Screen;
 5 import com.badlogic.gdx.audio.Sound;
 6 import com.badlogic.gdx.graphics.GL30;
 7 import com.badlogic.gdx.scenes.scene2d.Actor;
 8 import com.badlogic.gdx.scenes.scene2d.Stage;
 9 import com.badlogic.gdx.scenes.scene2d.ui.Image;
10 import com.badlogic.gdx.scenes.scene2d.ui.ImageButton;
11 import com.badlogic.gdx.scenes.scene2d.ui.Table;
12 import com.badlogic.gdx.scenes.scene2d.utils.ChangeListener;
13 import com.badlogic.gdx.scenes.scene2d.utils.Drawable;
14 import com.badlogic.gdx.scenes.scene2d.utils.TextureRegionDrawable;
15 import view.AssetManager;
16 import view.GdxGame;
17 import viewmodel.ScreenManager;
18
19 public class StartScreen implements Screen {
20     private GdxGame game;
21     private Stage stage;
22     private Sound clickSound;
23
24     public StartScreen(GdxGame gdxGame) {
25         this.game = gdxGame;
26         stage = new Stage(gdxGame.viewport, gdxGame.batch);
27         Gdx.input.setInputProcessor(stage);
28
29         setStage();
30     }
31
32     private void setStage() {
33         Table mainTable = new Table();
34         mainTable.setBounds(0, 0, GdxGame.virtualWidth, GdxGame.virtualHeight);
35         stage.addActor(mainTable);
36
37         clickSound = Gdx.audio.newSound(Gdx.files.internal("sounds/SFX/BigClick.mp3")
   );
38
39         Image background = new Image(AssetManager.getTextureRegion("
   StartScreenBackground"));
40         mainTable.setBackground(background.getDrawable());
41
42         Drawable drawable = new TextureRegionDrawable(AssetManager.getTextureRegion("
   TeacherButtonSkin"));
43         ImageButton teacherButton = new ImageButton(drawable);
44         teacherButton.addListener(new ChangeListener() {
45             @Override
46             public void changed(ChangeEvent event, Actor actor) {
47                 clickSound.play();
48                 ScreenManager.setScreen(new TeacherScreen(StartScreen.this.game));
49             }
50         });
```

```
 51
 52          drawable = new TextureRegionDrawable(AssetManager.getTextureRegion("
    StudentButtonSkin"));
 53          ImageButton studentButton = new ImageButton(drawable);
 54          studentButton.addListener(new ChangeListener() {
 55              @Override
 56              public void changed(ChangeEvent event, Actor actor) {
 57                  clickSound.play();
 58                  ScreenManager.setScreen(new StudentScreen(StartScreen.this.game));
 59              }
 60          });
 61
 62          mainTable.add().height(425);
 63          mainTable.row();
 64          mainTable.add(teacherButton).width(500).height(300);
 65          mainTable.add().width(350);
 66          mainTable.add(studentButton).width(500).height(300);
 67      }
 68
 69      @Override
 70      public void render(float delta) {
 71          Gdx.gl.glClearColor(0, 0, 0, 1);
 72          Gdx.gl.glClear(GL30.GL_COLOR_BUFFER_BIT);
 73          stage.act(delta);
 74          stage.draw();
 75      }
 76
 77      @Override
 78      public void resize(int width, int height) {
 79          stage.getViewport().update(width, height, false);
 80      }
 81
 82      @Override
 83      public void show() {
 84          Gdx.input.setInputProcessor(stage);
 85      }
 86
 87      @Override
 88      public void hide() {
 89      }
 90
 91      @Override
 92      public void pause() {
 93      }
 94
 95      @Override
 96      public void resume() {
 97      }
 98
 99      @Override
100      public void dispose() {
101          stage.dispose();
102      }
```

```
103 }
104
```

```java
package view.screens;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.Screen;
import com.badlogic.gdx.audio.Sound;
import com.badlogic.gdx.graphics.GL30;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.ui.Table;
import com.badlogic.gdx.scenes.scene2d.utils.ChangeListener;
import com.badlogic.gdx.scenes.scene2d.utils.TextureRegionDrawable;
import model.DataManager;
import view.AssetManager;
import view.GdxGame;
import view.games.SpellingGameScreen;
import viewmodel.ScreenManager;

/**
 * Screen for the student to use.
 */
public class StudentScreen implements Screen {
    private GdxGame game;
    private Stage stage;
    private Table mainTable;
    private Sound clickSound;

    public StudentScreen(GdxGame gdxGame) {
        this.game = gdxGame;
        stage = new Stage(gdxGame.viewport, gdxGame.batch);
        Gdx.input.setInputProcessor(stage);

        setStage();
    }

    private void setStage() {
        mainTable = new Table();
        mainTable.top().left().setBounds(0, 0, GdxGame.virtualWidth, GdxGame.
virtualHeight);
        mainTable.setBackground(new TextureRegionDrawable(AssetManager.
getTextureRegion("background")));
        stage.addActor(mainTable);

        clickSound = Gdx.audio.newSound(Gdx.files.internal("sounds/SFX/BigClick.mp3")
);

        selectTeachers();
    }

    /**
     * Shows all of the teachers.
     */
    private void selectTeachers() {
        String titleText = "Select a teacher to see their students.";
```

```
 51         if (DataManager.getTeachers().size() == 0) {
 52             titleText = "No teachers. Go back and create teacher.";
 53         }
 54         ChangeListener doOnBackButton = new ChangeListener() {
 55             @Override
 56             public void changed(ChangeEvent event, Actor actor) {
 57                 clickSound.play();
 58                 ScreenManager.setScreen(new StartScreen(StudentScreen.this.game));
 59             }
 60         };
 61         ChangeListener doAfterSelectItem = new ChangeListener() {
 62             @Override
 63             public void changed(ChangeEvent event, Actor actor) {
 64                 clickSound.play();
 65                 StudentScreen.this.selectStudents();
 66             }
 67         };
 68 
 69         mainTable.clearChildren();
 70         mainTable.addActor(ScreenManager.screenFactory(ScreenManager.ScreenType.
    TEACHERS, titleText, doOnBackButton, doAfterSelectItem, null, null));
 71     }
 72 
 73     /**
 74      * Shows the students under the selected teacher.
 75      */
 76     private void selectStudents() {
 77         String titleText = "Select your name and let's play a game!";
 78         if (ScreenManager.getSelectedTeacher().getStudents().size() == 0) {
 79             titleText = "No students. Go back and add a student.";
 80         }
 81         ChangeListener doOnBackButton = new ChangeListener() {
 82             @Override
 83             public void changed(ChangeEvent event, Actor actor) {
 84                 clickSound.play();
 85                 selectTeachers();
 86             }
 87         };
 88         ChangeListener doAfterSelectItem = new ChangeListener() {
 89             @Override
 90             public void changed(ChangeEvent event, Actor actor) {
 91                 clickSound.play();
 92                 StudentScreen.this.displayGames();
 93             }
 94         };
 95 
 96         mainTable.clearChildren();
 97         mainTable.addActor(ScreenManager.screenFactory(ScreenManager.ScreenType.
    STUDENTS, titleText, doOnBackButton, doAfterSelectItem, null, null));
 98     }
 99 
100     /**
101      * Shows the games and languages available.
```

```
102      */
103     private void displayGames() {
104         String titleText = "Select a language and a game.";
105         ChangeListener doOnBackButton = new ChangeListener() {
106             @Override
107             public void changed(ChangeEvent event, Actor actor) {
108                 clickSound.play();
109                 selectStudents();
110             }
111         };
112         ChangeListener doAfterSelectItem = new ChangeListener() {
113             @Override
114             public void changed(ChangeEvent event, Actor actor) {
115                 clickSound.play();
116                 clickSound = Gdx.audio.newSound(Gdx.files.internal("sounds/SFX/cheer
    .mp3"));
117                 clickSound.play();
118                 ScreenManager.setPreviousScreen(StudentScreen.this);
119                 ScreenManager.setScreen(new SpellingGameScreen(StudentScreen.this.
    game));
120             }
121         };
122 
123         mainTable.clearChildren();
124         mainTable.addActor(ScreenManager.screenFactory(ScreenManager.ScreenType.
    GAMES, titleText, doOnBackButton, doAfterSelectItem, null, null));
125     }
126 
127     @Override
128     public void render(float delta) {
129         Gdx.gl.glClearColor(0, 0, 0, 1);
130         Gdx.gl.glClear(GL30.GL_COLOR_BUFFER_BIT);
131         stage.act(delta);
132         stage.draw();
133     }
134 
135     @Override
136     public void resize(int width, int height) {
137         stage.getViewport().update(width, height, false);
138     }
139 
140     @Override
141     public void show() {
142         Gdx.input.setInputProcessor(stage);
143     }
144 
145     @Override
146     public void hide() {
147     }
148 
149     @Override
150     public void pause() {
151     }
```

```
152
153     @Override
154     public void resume() {
155     }
156
157     @Override
158     public void dispose() {
159         clickSound.dispose();
160         stage.dispose();
161     }
162 }
163
```

```java
package view.screens;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.Screen;
import com.badlogic.gdx.audio.Sound;
import com.badlogic.gdx.graphics.GL30;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.ui.Table;
import com.badlogic.gdx.scenes.scene2d.utils.ChangeListener;
import com.badlogic.gdx.scenes.scene2d.utils.TextureRegionDrawable;
import model.DataManager;
import view.AssetManager;
import view.GdxGame;
import viewmodel.ScreenManager;

/**
 * Screen for the teacher to use.
 */
public class TeacherScreen implements Screen {
    private GdxGame game;
    public static Stage stage;
    private Table mainTable;
    private Sound clickSound;

    public TeacherScreen(GdxGame gdxGame) {
        this.game = gdxGame;
        stage = new Stage(gdxGame.viewport, gdxGame.batch);
        Gdx.input.setInputProcessor(stage);

        setStage();
    }

    private void setStage() {
        mainTable = new Table();
        mainTable.top().left().setBounds(0, 0, GdxGame.virtualWidth, GdxGame.
virtualHeight);
        mainTable.setBackground(new TextureRegionDrawable(AssetManager.
getTextureRegion("background")));
        stage.addActor(mainTable);

        clickSound = Gdx.audio.newSound(Gdx.files.internal("sounds/SFX/BigClick.mp3")
);

        selectTeachers();
    }

    /**
     * Shows all of the teachers.
     */
    private void selectTeachers() {
        String titleText = "Select a teacher to see their students.";
        if (DataManager.getTeachers().size() == 0) {
```

```
51              titleText = "Add a teacher to see their students.";
52          }
53          ChangeListener doOnBackButton = new ChangeListener() {
54              @Override
55              public void changed(ChangeEvent event, Actor actor) {
56                  clickSound.play();
57                  ScreenManager.setScreen(new StartScreen(TeacherScreen.this.game));
58              }
59          };
60          ChangeListener doAfterSelectItem = new ChangeListener() {
61              @Override
62              public void changed(ChangeEvent event, Actor actor) {
63                  clickSound.play();
64                  TeacherScreen.this.selectStudents();
65              }
66          };
67
68          String addItemInfoText = "<Teacher name>";
69          ChangeListener doAfterAddRemove = new ChangeListener() {
70              @Override
71              public void changed(ChangeEvent event, Actor actor) {
72                  TeacherScreen.this.selectTeachers();
73              }
74          };
75
76          mainTable.clearChildren();
77          mainTable.addActor(ScreenManager.screenFactory(ScreenManager.ScreenType.
    TEACHERS, titleText, doOnBackButton, doAfterSelectItem, addItemInfoText,
    doAfterAddRemove));
78      }
79
80      /**
81       * Shows the students under the selected teacher.
82       */
83      private void selectStudents() {
84          String titleText = "Select a student to see their history.";
85          if (ScreenManager.getSelectedTeacher().getStudents().size() == 0) {
86              titleText = "No students. Add a student.";
87          }
88          ChangeListener doOnBackButton = new ChangeListener() {
89              @Override
90              public void changed(ChangeEvent event, Actor actor) {
91                  clickSound.play();
92                  TeacherScreen.this.selectTeachers();
93              }
94          };
95          ChangeListener doAfterSelectItem = new ChangeListener() {
96              @Override
97              public void changed(ChangeEvent event, Actor actor) {
98                  clickSound.play();
99                  displayHistories();
100             }
101         };
```

```
102
103         String addItemInfoText = "<Student name>";
104         ChangeListener doAfterAddRemove = new ChangeListener() {
105             @Override
106             public void changed(ChangeEvent event, Actor actor) {
107                 clickSound.play();
108                 TeacherScreen.this.selectStudents();
109             }
110         };
111
112         mainTable.clearChildren();
113         mainTable.addActor(ScreenManager.screenFactory(ScreenManager.ScreenType.
    STUDENTS, titleText, doOnBackButton, doAfterSelectItem, addItemInfoText,
    doAfterAddRemove));
114     }
115
116     /**
117      * Shows histories of the selected student.
118      */
119     private void displayHistories() {
120         String titleText = "Student's game history:";
121         ChangeListener doOnBackButton = new ChangeListener() {
122             @Override
123             public void changed(ChangeEvent event, Actor actor) {
124                 clickSound.play();
125                 TeacherScreen.this.selectStudents();
126             }
127         };
128
129         mainTable.clearChildren();
130         mainTable.addActor(ScreenManager.screenFactory(ScreenManager.ScreenType.
    HISTORIES, titleText, doOnBackButton, null, null, null));
131     }
132
133     @Override
134     public void render(float delta) {
135         Gdx.gl.glClearColor(0, 0, 0, 1);
136         Gdx.gl.glClear(GL30.GL_COLOR_BUFFER_BIT);
137         stage.act(delta);
138         stage.draw();
139     }
140
141     @Override
142     public void resize(int width, int height) {
143         stage.getViewport().update(width, height, false);
144     }
145
146     @Override
147     public void show() {
148         Gdx.input.setInputProcessor(stage);
149     }
150
151     @Override
```

```
152     public void hide() {
153     }
154
155     @Override
156     public void pause() {
157     }
158
159     @Override
160     public void resume() {
161     }
162
163     @Override
164     public void dispose() {
165         clickSound.dispose();
166         stage.dispose();
167     }
168 }
169
```

File - C:\Users\anon\Desktop\hmonics\core\src\view\screens\LibGDXSplashScreen.java

```java
package view.screens;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.Screen;
import com.badlogic.gdx.audio.Sound;
import com.badlogic.gdx.graphics.GL30;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.actions.Actions;
import com.badlogic.gdx.scenes.scene2d.ui.Image;
import view.AssetManager;
import view.GdxGame;
import viewmodel.ScreenManager;

public class LibGDXSplashScreen implements Screen {
    private GdxGame game;
    private Stage stage;
    private Sound giggles;

    public LibGDXSplashScreen(GdxGame gdxGame) {
        this.game = gdxGame;
        stage = new Stage(gdxGame.viewport, gdxGame.batch);
        Gdx.input.setInputProcessor(stage);

        setStage();
    }

    private void setStage() {
        Image image = new Image(AssetManager.getTextureRegion("libGDXSplash"));
        image.setSize(GdxGame.virtualWidth, GdxGame.virtualHeight);
        stage.addActor(image);

        giggles = Gdx.audio.newSound(Gdx.files.internal("sounds/SFX/giggles.mp3"));
        giggles.play();

        image.addAction(Actions.sequence(
                Actions.delay(2f),
                Actions.run(new Runnable() {
                    public void run() {
                        ScreenManager.start(game, new StartScreen(game));
                    }
                })
        ));
    }

    @Override
    public void render(float delta) {
        Gdx.gl.glClearColor(0, 0, 0, 1);
        Gdx.gl.glClear(GL30.GL_COLOR_BUFFER_BIT);
        stage.act(delta);
        stage.draw();
    }

    @Override
```

```
54      public void resize(int width, int height) {
55          stage.getViewport().update(width, height, false);
56      }
57
58      @Override
59      public void show() {
60          Gdx.input.setInputProcessor(stage);
61      }
62
63      @Override
64      public void hide() {
65      }
66
67      @Override
68      public void pause() {
69      }
70
71      @Override
72      public void resume() {
73      }
74
75      @Override
76      public void dispose() {
77          stage.dispose();
78      }
79  }
80
81
```

File - C:\Users\anon\Desktop\hmonics\core\src\view\screens\TeamLogoSplashScreen.java

```java
package view.screens;

import com.badlogic.gdx.Gdx;
import com.badlogic.gdx.Screen;
import com.badlogic.gdx.audio.Music;
import com.badlogic.gdx.graphics.GL30;
import com.badlogic.gdx.scenes.scene2d.Stage;
import com.badlogic.gdx.scenes.scene2d.actions.Actions;
import com.badlogic.gdx.scenes.scene2d.ui.Image;
import view.AssetManager;
import view.GdxGame;
import viewmodel.ScreenManager;

public class TeamLogoSplashScreen implements Screen {
    private GdxGame game;
    private Stage stage;
    private static Music backgroundMusic;

    public TeamLogoSplashScreen(GdxGame gdxGame) {
        this.game = gdxGame;
        stage = new Stage(gdxGame.viewport, gdxGame.batch);
        Gdx.input.setInputProcessor(stage);

        setStage();
    }

    private void setStage() {
        Image image = new Image(AssetManager.getTextureRegion("TeamLogoSplash"));
        image.setSize(GdxGame.virtualWidth, GdxGame.virtualHeight);
        stage.addActor(image);

        backgroundMusic = AssetManager.getMusic("IntroMusic");
        backgroundMusic.play();

        image.addAction(Actions.sequence(
                Actions.delay(2f),
                Actions.run(new Runnable() {
                    public void run() {
                        ScreenManager.start(game, new LibGDXSplashScreen(game));
                    }
                })
        ));
    }

    @Override
    public void render(float delta) {
        Gdx.gl.glClearColor(0, 0, 0, 1);
        Gdx.gl.glClear(GL30.GL_COLOR_BUFFER_BIT);
        stage.act(delta);
        stage.draw();
    }

    @Override
```

File - C:\Users\anon\Desktop\hmonics\core\src\view\screens\TeamLogoSplashScreen.java

```
54      public void resize(int width, int height) {
55          stage.getViewport().update(width, height, false);
56      }
57
58      @Override
59      public void show() {
60          Gdx.input.setInputProcessor(stage);
61      }
62
63      public static Music getBackgroundMusic() {
64          return backgroundMusic;
65      }
66
67      @Override
68      public void hide() {
69      }
70
71      @Override
72      public void pause() {
73      }
74
75      @Override
76      public void resume() {
77      }
78
79      @Override
80      public void dispose() {
81          stage.dispose();
82      }
83  }
84
```

File - C:\Users\anon\Desktop\hmonics\core\src\model\Word.java

```java
package model;

import viewmodel.ScreenManager.Language;

/**
 * A Word has an english and hmong spelling. The ID associated with a Word is the english spelling. A hmong word has
 * a certain amount of game spaces that differ from character count.
 */
public class Word implements Comparable<String> {
    private String englishSpelling;
    private String hmongSpelling;
    private int hmongSpaceLength;

    public Word(String englishSpelling, String hmongSpelling, int hmongSpaceLength) {
        this.englishSpelling = englishSpelling;
        this.hmongSpelling = hmongSpelling;
        this.hmongSpaceLength = hmongSpaceLength;
    }

    public String getWordId() {
        return englishSpelling;
    }

    /**
     * Simply returns a string based on what language
     */
    public String getSpelling(Language language) {
        switch (language) {
            case ENGLISH:
                return englishSpelling;
            case HMONG:
                return hmongSpelling;
            default:
                return null;
        }
    }

    /**
     * Returns the number of spaces based on language.
     *
     * @return
     */
    public int getSpaceLength(Language language) {
        switch (language) {
            case ENGLISH:
                return englishSpelling.length();
            case HMONG:
                return hmongSpaceLength;
            default:
                return 0;
        }
    }
```

```
53
54      public void setHmongSpaceLength(int hmongSpaceLength) {
55          this.hmongSpaceLength = hmongSpaceLength;
56      }
57
58      @Override
59      public int compareTo(String o) {
60          return o.compareTo(this.englishSpelling);
61      }
62  }
63
```

File - C:\Users\anon\Desktop\hmonics\core\src\model\History.java

```java
package model;

import java.text.SimpleDateFormat;
import java.util.ArrayList;
import java.util.Date;
/**
 * Each Student creates a History object after every game played.
 */
public class History {
    private String gamePlayed;
    private ArrayList<String> wordsSpelled;
    private Date date;
    private String timestamp;

    public History(String gamePlayed) {
        this.gamePlayed = gamePlayed;
        wordsSpelled = new ArrayList<String>();
        SimpleDateFormat dateFormat = new SimpleDateFormat("MM/dd HH:mm");
        date = new Date();
        timestamp = dateFormat.format(date);
    }

    public String getGamePlayed() {
        return gamePlayed;
    }

    public ArrayList<String> getWordsSpelled() {
        return wordsSpelled;
    }

    public Date getDate() {
        return date;
    }

    public String getDateString() {
        return timestamp;
    }

    public void addWord(String word) {
        wordsSpelled.add(word);
    }
}

```

File - C:\Users\anon\Desktop\hmonics\core\src\model\Student.java

```java
package model;

import java.util.ArrayList;

public class Student {
    private String name;
    private ArrayList<History> gameHistories;
    private History currentHistory;

    public Student(String name) {
        this.name = name;
        gameHistories = new ArrayList<History>();
    }

    public String getName() {
        return name;
    }

    public ArrayList<History> getGameHistory() {
        return gameHistories;
    }

    public void startNewCurrentHistory(History history) {
        currentHistory = history;
        gameHistories.add(history);
    }

    public void addToCurrentHistory(String word) {
        currentHistory.addWord(word);
    }
}

```

File - C:\Users\anon\Desktop\hmonics\core\src\model\Teacher.java

```java
package model;

import java.util.ArrayList;

public class Teacher {
    private String name;
    private ArrayList<Student> students;

    public Teacher(String name) {
        this.name = name;
        students = new ArrayList<Student>();
    }

    public String getName() {
        return name;
    }

    public ArrayList<Student> getStudents() {
        return students;
    }
}

```

File - C:\Users\anon\Desktop\hmonics\core\src\model\DataManager.java

```java
package model;

import java.util.ArrayList;
import java.util.Collections;
import java.util.Comparator;

/**
 * Handles Teachers, Students, and Words storage.
 */
public class DataManager {
    private static ArrayList<Teacher> teachers = new ArrayList<Teacher>();
    private static ArrayList<Word> wordList = new ArrayList<Word>();

    public static void populate() {
        wordList.add(new Word("apple", "kua", 2));
        wordList.add(new Word("money", "nyiaj", 3));
        wordList.add(new Word("bird", "noog", 3));
        wordList.add(new Word("pig", "npua", 2));
        wordList.add(new Word("dog", "aub", 2));
        wordList.add(new Word("boat", "nkoj", 3));
        wordList.add(new Word("fish", "ntses", 3));
        wordList.add(new Word("deer", "mos lwj", 7));
        wordList.add(new Word("cat", "miv", 3));
        wordList.add(new Word("horse", "nees", 3));
        wordList.add(new Word("flower", "paj", 3));
        wordList.add(new Word("frog", "qav", 3));
        wordList.add(new Word("pumpkin", "taub dag", 7));
        wordList.add(new Word("sheep", "yaj", 3));
        wordList.add(new Word("dragon", "zaj", 3));

        // TESTING
//        addTeacher(new Teacher("Mrs. Anderson"));
//        addTeacher(new Teacher("Mr. Johnson"));
//        addTeacher(new Teacher("Mrs. Smith"));
//
//        addStudent(1, new Student("Jared"));
//        addStudent(1, new Student("Derick"));
//        addStudent(1, new Student("Philip"));
//
//        for (int i = 0; i < 9; i++) {
//            Student student;
//            addStudent(0, student = new Student("Student " + (i + 1)));
//            History history;
//            student.startNewCurrentHistory(history = new History("Spelling Game"));
//            history.addWord("bird");
//            history.addWord("cat");
//            history.addWord("dog");
//        }
    }

    public static ArrayList<Word> getWordList() {
        return wordList;
    }
```

```
54
55      public static Word getWord(String wordId) {
56          for (Word word : wordList) {
57              if (word.compareTo(wordId) == 0)
58                  return word;
59          }
60          return null;
61      }
62
63      public static ArrayList<Teacher> getTeachers() {
64          Collections.sort(teachers, new Comparator<Teacher>() {
65              @Override
66              public int compare(Teacher o1, Teacher o2) {
67                  return o1.getName().compareToIgnoreCase(o2.getName());
68              }
69          });
70          return teachers;
71      }
72
73      public static ArrayList<Student> getStudents(int teacherIndex) {
74          Collections.sort(teachers.get(teacherIndex).getStudents(), new Comparator<
    Student>() {
75              @Override
76              public int compare(Student o1, Student o2) {
77                  return o1.getName().compareToIgnoreCase(o2.getName());
78              }
79          });
80          return teachers.get(teacherIndex).getStudents();
81      }
82
83      public static ArrayList<History> getHistory(Student student) {
84          Collections.sort(student.getGameHistory(), new Comparator<History>() {
85              @Override
86              public int compare(History o1, History o2) {
87                  return o2.getDate().compareTo(o1.getDate());
88              }
89          });
90          return student.getGameHistory();
91      }
92
93      public static void addTeacher(Teacher teacher) {
94          teachers.add(teacher);
95      }
96
97      public static void removeTeacher(int teacherIndex) {
98          teachers.remove(teacherIndex);
99      }
100
101     public static void addStudent(int teacherIndex, Student student) {
102         teachers.get(teacherIndex).getStudents().add(student);
103     }
104
105     public static void removeStudent(int teacherIndex, int studentIndex) {
```

File - C:\Users\anon\Desktop\hmonics\core\src\model\DataManager.java

```
106             teachers.get(teacherIndex).getStudents().remove(studentIndex);
107         }
108 }
109
```

File - C:\Users\anon\Desktop\hmonics\core\src\viewmodel\ScreenManager.java

```java
package viewmodel;

import com.badlogic.gdx.Screen;
import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.InputEvent;
import com.badlogic.gdx.scenes.scene2d.ui.*;
import com.badlogic.gdx.scenes.scene2d.utils.ChangeListener;
import com.badlogic.gdx.scenes.scene2d.utils.ClickListener;
import com.badlogic.gdx.utils.Align;
import model.DataManager;
import model.History;
import model.Student;
import model.Teacher;
import view.AssetManager;
import view.GdxGame;
import view.screens.TeacherScreen;

import java.util.ArrayList;

/**
 * Creates screens and manages the global state variables.
 * Stores a previous screen when appropriate.
 */
public class ScreenManager {
    public static Language selectedLanguage;
    public static int selectedTeacherIndex;
    public static String selectedTeacherName;
    public static int selectedStudentIndex;
    public static String selectedStudentName;

    private static GdxGame game;
    private static Screen previousScreen;

    public static void start(GdxGame gdxGame, Screen firstScreen) {
        ScreenManager.game = gdxGame;
        selectedLanguage = Language.HMONG;
        setScreen(firstScreen);
    }

    public static void setScreen(Screen next) {
        game.setScreen(next);
    }

    public static void setPreviousScreen(Screen previousScreen) {
        ScreenManager.previousScreen = previousScreen;
    }

    public static Screen getPreviousScreen() {
        return previousScreen;
    }

    public static Teacher getSelectedTeacher() {
        return DataManager.getTeachers().get(selectedTeacherIndex);
```

```
54      }
55
56      public static Student getSelectedStudent() {
57          return DataManager.getStudents(ScreenManager.selectedTeacherIndex).get(
    ScreenManager.selectedStudentIndex);
58      }
59
60      /**
61       * Creates a table that is all parts of the non-game screens.
62       * The name of each button is the list index for that button's function.
63       *
64       * @param screenType        What kind of screen.
65       * @param titleText         Text for the title of the screen.
66       * @param doOnBackButton    Do this after back button is pressed.
67       * @param doAfterSelectItem Do this after an item is selected.
68       * @param addItemInfoText   Info text for the default name in the add item field
    .
69       * @param doAfterAddRemove  Do this after an item has been added or removed.
70       * @return A table to be used as a screen.
71       */
72      public static Table screenFactory(ScreenType screenType, String titleText,
    ChangeListener doOnBackButton, ChangeListener doAfterSelectItem, String
    addItemInfoText, ChangeListener doAfterAddRemove) {
73          int backButtonSize = 150;
74
75          Table mainTable = new Table();
76          mainTable.setBounds(0, 0, GdxGame.virtualWidth, GdxGame.virtualHeight);
77          Label titleLabel = new Label(titleText, AssetManager.labelStyle64Clear);
78          titleLabel.setBounds(300, mainTable.getHeight() - 220, mainTable.getWidth()
    - 600, 200);
79          mainTable.addActor(titleLabel);
80          Table bodyTable = new Table();
81          bodyTable.setBounds(300, mainTable.getHeight() - 750 - 250, mainTable.
    getWidth() - 600, 750);
82          mainTable.addActor(bodyTable);
83          VerticalGroup verticalGroup = new VerticalGroup();
84          verticalGroup.align(Align.topLeft);
85          ScrollPane scrollPane = new ScrollPane(verticalGroup, AssetManager.
    defaultSkin);
86          scrollPane.setScrollingDisabled(true, false);
87          scrollPane.setFadeScrollBars(false);
88
89          ImageButton backButton = new ImageButton(AssetManager.backButtonStyle);
90          backButton.setBounds(50, 50, backButtonSize, backButtonSize);
91          backButton.setSize(backButtonSize, backButtonSize);
92          backButton.addListener(doOnBackButton);
93          mainTable.addActor(backButton);
94
95          int rowHeight = 100;
96          int rowSeparator = 25;
97          Table aDataRow = new Table();
98
99          switch (screenType) {
```

```
100             case TEACHERS:
101             case STUDENTS:
102                 boolean allowChanges = !(addItemInfoText == null);
103                 int nameWidth = 800;
104                 int columnSeparator = 75;
105                 int buttonWidth = 300;
106 
107                 // If the screen allows changes to entries, have first row be the
    add entry row.
108                 if (allowChanges) {
109                     aDataRow.background(AssetManager.backPlate);
110                     aDataRow.pad(10);
111 
112                     final TextField addItemField = new TextField(addItemInfoText,
    AssetManager.textFieldStyle64);
113                     addItemField.addListener(new ClickListener() {
114                         boolean firstClick = true;
115 
116                         @Override
117                         public void clicked(InputEvent event, float x, float y) {
118                             super.clicked(event, x, y);
119                             if (firstClick) addItemField.selectAll();
120                             firstClick = false;
121                         }
122                     });
123 
124                     // Button will add a new teacher or student with the string in
    the text field as the name.
125                     TextButton addButton = new TextButton("Add", AssetManager.
    textButtonStyle64);
126                     switch (screenType) {
127                         case TEACHERS:
128                             addButton.addListener(new ChangeListener() {
129                                 @Override
130                                 public void changed(ChangeEvent event, Actor actor)
    {
131                                     DataManager.addTeacher(new Teacher(addItemField.
    getText()));
132                                 }
133                             });
134                             break;
135                         case STUDENTS:
136                             addButton.addListener(new ChangeListener() {
137                                 @Override
138                                 public void changed(ChangeEvent event, Actor actor)
    {
139                                     DataManager.addStudent(selectedTeacherIndex, new
     Student(addItemField.getText()));
140                                 }
141                             });
142                             break;
143                     }
144                     addButton.addListener(doAfterAddRemove);
```

```
145
146                     aDataRow.add(addItemField).width(nameWidth).height(rowHeight).
    left();
147                     aDataRow.add().width(columnSeparator);
148                     aDataRow.add(addButton).width(buttonWidth).height(rowHeight);
149                     bodyTable.add(aDataRow).left();
150                     bodyTable.row();
151                 }
152
153                 // Make a row for each entry of either teachers or students.
154                 switch (screenType) {
155                     case TEACHERS:
156                         final ArrayList<Teacher> teachers = DataManager.getTeachers(
    );
157                         for (int i = 0; i < teachers.size(); i++) { // Make a data
    row for each teacher.
158                             aDataRow = new Table();
159
160                             // Button for selecting a teacher.
161                             TextButton nameButton = new TextButton(teachers.get(i).
    getName(), AssetManager.textButtonStyle64);
162                             nameButton.setName(String.valueOf(i));
163                             nameButton.getLabel().setAlignment(Align.left);
164                             nameButton.addListener(new ChangeListener() {
165                                 @Override
166                                 public void changed(ChangeEvent event, Actor actor)
    {
167                                     selectedTeacherIndex = Integer.parseInt(actor.
    getName());
168                                     selectedTeacherName = DataManager.getTeachers().
    get(selectedTeacherIndex).getName();
169                                 }
170                             });
171                             nameButton.addListener(doAfterSelectItem);
172                             aDataRow.add(nameButton).width(nameWidth).height(
    rowHeight);
173
174                             // Button for removing a teacher.
175                             if (allowChanges) {
176                                 TextButton deleteButton = new TextButton("Delete",
    AssetManager.textButtonStyle64);
177                                 deleteButton.setName(String.valueOf(i));
178                                 deleteButton.addListener(new ChangeListener() {
179                                     @Override
180                                     public void changed(ChangeEvent event, Actor
    actor) {
181                                         DataManager.removeTeacher(Integer.parseInt(
    actor.getName()));
182                                     }
183                                 });
184                                 deleteButton.addListener(doAfterAddRemove);
185                                 aDataRow.add().width(columnSeparator);
186                                 aDataRow.add(deleteButton).width(buttonWidth).height
```

```
186 (rowHeight);
187                             }
188 
189                             aDataRow.row();
190                             aDataRow.add().height(rowSeparator);
191                             verticalGroup.addActor(aDataRow);
192                         }
193                         break;
194                     case STUDENTS:
195                         final ArrayList<Student> students = DataManager.getStudents(
    selectedTeacherIndex);
196                         for (int i = 0; i < students.size(); i++) { // Make a data
    row for each student.
197                             aDataRow = new Table();
198 
199                             // Button for selecting a student.
200                             TextButton nameButton = new TextButton(students.get(i).
    getName(), AssetManager.textButtonStyle64);
201                             nameButton.setName(String.valueOf(i));
202                             nameButton.getLabel().setAlignment(Align.left);
203                             nameButton.addListener(new ChangeListener() {
204                                 @Override
205                                 public void changed(ChangeEvent event, Actor actor)
    {
206                                     selectedStudentIndex = Integer.parseInt(actor.
    getName());
207                                     selectedStudentName = DataManager.getStudents(
    selectedTeacherIndex).get(selectedStudentIndex).getName();
208                                 }
209                             });
210                             nameButton.addListener(doAfterSelectItem);
211                             aDataRow.add(nameButton).width(nameWidth).height(
    rowHeight);
212 
213                             // Button for removing a student.
214                             if (allowChanges) {
215                                 TextButton deleteButton = new TextButton("Delete",
    AssetManager.textButtonStyle64);
216                                 deleteButton.setName(String.valueOf(i));
217                                 deleteButton.addListener(new ChangeListener() {
218                                     @Override
219                                     public void changed(ChangeEvent event, Actor
    actor) {
220                                         DataManager.removeStudent(
    selectedTeacherIndex, Integer.parseInt(actor.getName()));
221                                     }
222                                 });
223                                 deleteButton.addListener(doAfterAddRemove);
224                                 aDataRow.add().width(columnSeparator);
225                                 aDataRow.add(deleteButton).width(buttonWidth).height
    (rowHeight);
226                             }
227 
```

```
228                             aDataRow.row();
229                             aDataRow.add().height(rowSeparator);
230                             verticalGroup.addActor(aDataRow);
231                         }
232                         break;
233                 }
234 
235                 Table backgroundTable = new Table();
236                 backgroundTable.background(AssetManager.backPlate);
237                 backgroundTable.pad(10);
238                 bodyTable.add(backgroundTable);
239                 if (allowChanges) {
240                     backgroundTable.add(scrollPane).width(nameWidth +
    columnSeparator + buttonWidth + 20).height((rowHeight + rowSeparator) * 5);
241                 } else {
242                     backgroundTable.add(scrollPane).width(nameWidth + 20).height((
    rowHeight + rowSeparator) * 6);
243                 }
244                 break;
245             case HISTORIES:
246                 int dateWidth = 400;
247                 int gameNameWidth = 500;
248                 int numberWidth = 150;
249                 columnSeparator = 50;
250 
251                 final ArrayList<History> histories = DataManager.getHistory(
    DataManager.getStudents(selectedTeacherIndex).get(selectedStudentIndex));
252                 for (int i = 0; i < histories.size(); i++) { // Make a data row for
    each history.
253                     aDataRow = new Table();
254                     Label dateLabel = new Label(String.valueOf(histories.get(i).
    getDateString()), AssetManager.labelStyle64Clear);
255                     dateLabel.setAlignment(Align.center);
256                     Label gameNameLabel = new Label(histories.get(i).getGamePlayed()
    , AssetManager.labelStyle64Clear);
257                     gameNameLabel.setAlignment(Align.center);
258 
259                     // Button that shows a dialog of what words were spelled.
260                     TextButton numberOfWordsButton = new TextButton(String.valueOf(
    histories.get(i).getWordsSpelled().size()), AssetManager.textButtonStyle64);
261                     numberOfWordsButton.setName(String.valueOf(i));
262                     numberOfWordsButton.addListener(new ChangeListener() {
263                         @Override
264                         public void changed(ChangeEvent event, Actor actor) {
265                             StringBuilder wordList = new StringBuilder();
266                             wordList.append("Words Spelled:\n");
267                             ArrayList<String> wordsSpelled = histories.get(Integer.
    parseInt(actor.getName())).getWordsSpelled();
268                             for (int j = 1; j <= wordsSpelled.size(); j++) {
269                                 wordList.append(wordsSpelled.get(j - 1));
270                                 if (j != wordsSpelled.size()) wordList.append(", ");
     // last item, do not add a comma
271                                 if (j % 3 == 0) wordList.append("\n"); // every
```

```
271 third, add a new line
272                             }
273                             Dialog dialog = new Dialog("", AssetManager.defaultSkin)
    ;
274                             dialog.getBackground().setMinWidth(500);
275                             dialog.getBackground().setMinHeight(300);
276                             Label label = new Label(wordList.toString(),
    AssetManager.labelStyle64Solid);
277                             label.setAlignment(Align.center);
278                             dialog.text(label);
279                             dialog.button(new TextButton("OK", AssetManager.
    textButtonStyle64));
280                             dialog.show(TeacherScreen.stage);
281                         }
282                     });
283 
284                     aDataRow.add(dateLabel).width(dateWidth).height(rowHeight);
285                     aDataRow.add().width(columnSeparator);
286                     aDataRow.add(gameNameLabel).width(gameNameWidth).height(
    rowHeight);
287                     aDataRow.add().width(columnSeparator);
288                     aDataRow.add(numberOfWordsButton).width(numberWidth).height(
    rowHeight);
289                     aDataRow.row();
290                     aDataRow.add().height(rowSeparator);
291                     verticalGroup.addActor(aDataRow);
292                 }
293 
294                 backgroundTable = new Table();
295                 backgroundTable.background(AssetManager.backPlate);
296                 backgroundTable.pad(10);
297                 bodyTable.add(backgroundTable);
298                 backgroundTable.add(scrollPane).width(dateWidth + gameNameWidth +
    numberWidth + columnSeparator * 2).height((rowHeight + rowSeparator) * 6);
299                 break;
300             case GAMES:
301                 gameNameWidth = 500;
302                 int gameNameHeight = 200;
303                 columnSeparator = 100;
304 
305                 bodyTable.background(AssetManager.backPlate);
306                 Table gamesList = new Table();
307                 bodyTable.add(gamesList);
308                 bodyTable.add().width(columnSeparator);
309 
310                 TextButton spellingGameButton = new TextButton("Spelling Game",
    AssetManager.textButtonStyle64);
311                 spellingGameButton.addListener(doAfterSelectItem);
312                 gamesList.add(spellingGameButton).width(gameNameWidth).height(
    gameNameHeight);
313 
314                 Table languageSelectList = new Table();
315                 bodyTable.add(languageSelectList);
```

```
316
317                 // Button to select English as the language for the game.
318                 TextButton englishButton = new TextButton("English", AssetManager.
    textButtonStyle64Checked);
319                 englishButton.addListener(new ChangeListener() {
320                     @Override
321                     public void changed(ChangeEvent event, Actor actor) {
322                         ScreenManager.selectedLanguage = Language.ENGLISH;
323                     }
324                 });
325                 languageSelectList.add(englishButton).width(gameNameWidth).height(
    gameNameHeight);
326                 languageSelectList.row();
327
328                 // Button to select Hmong as the language for the game.
329                 TextButton hmongButton = new TextButton("Hmong", AssetManager.
    textButtonStyle64Checked);
330                 hmongButton.addListener(new ChangeListener() {
331                     @Override
332                     public void changed(ChangeEvent event, Actor actor) {
333                         ScreenManager.selectedLanguage = Language.HMONG;
334                     }
335                 });
336                 languageSelectList.add(hmongButton).width(gameNameWidth).height(
    gameNameHeight);
337
338                 switch (selectedLanguage) {
339                     case ENGLISH:
340                         englishButton.setChecked(true);
341                         break;
342                     case HMONG:
343                         hmongButton.setChecked(true);
344                         break;
345                 }
346                 ButtonGroup<TextButton> buttonGroup = new ButtonGroup<TextButton>();
347                 buttonGroup.setMaxCheckCount(1);
348                 buttonGroup.setMinCheckCount(1);
349                 buttonGroup.add(englishButton);
350                 buttonGroup.add(hmongButton);
351                 break;
352         }
353         return mainTable;
354     }
355
356     /**
357      * Screen types that can be created by screenFactory.
358      */
359     public enum ScreenType {
360         TEACHERS, STUDENTS, HISTORIES, GAMES
361     }
362
363     /**
364      * Currently supported languages.
```

```
365       */
366      public enum Language {
367          ENGLISH("English"), HMONG("Hmong");
368          public final String fileName;
369
370          Language(String fileName) {
371              this.fileName = fileName;
372          }
373      }
374  }
375
```

```java
package viewmodel;

import com.badlogic.gdx.scenes.scene2d.Actor;
import com.badlogic.gdx.scenes.scene2d.InputEvent;
import com.badlogic.gdx.scenes.scene2d.actions.Actions;
import model.DataManager;
import model.History;
import model.Student;
import model.Word;
import view.actors.Letter;
import view.games.SpellingGameScreen;

import java.util.ArrayList;
import java.util.Random;

/**
 * This class handles each letter drop in the Spelling Game.
 */
public class SpellingGameManager {
    private Random random;
    private SpellingGameScreen spellingGameScreen;
    private ScreenManager.Language currentLanguage;
    private Student currentStudent;
    private ArrayList<Word> sessionWordList;
    private Word currentWord;

    public SpellingGameManager(SpellingGameScreen spellingGameScreen) {
        this.random = new Random(System.currentTimeMillis());
        this.spellingGameScreen = spellingGameScreen;

        this.currentLanguage = ScreenManager.selectedLanguage;
        spellingGameScreen.setDisplayLanguage(currentLanguage);

        this.currentStudent = ScreenManager.getSelectedStudent();
        this.currentStudent.startNewCurrentHistory(new History("Spelling Game"));

        this.sessionWordList = new ArrayList<Word>(DataManager.getWordList());
        changeToNextWord();
    }

    public void droppedLetter(final Actor actor) {
        spellingGameScreen.playSFX("LetterClick");
        System.out.println("Current word: " + spellingGameScreen.getWordInSpaces());
        if (wordIsCorrect()) {
            recordWord();
            actor.addAction(Actions.sequence(
                    Actions.run(new Runnable() {
                        public void run() {
                            spellingGameScreen.playLetter(currentLanguage.fileName, (
Letter) actor);
                        }
                    }),
                    Actions.delay(0.5f),
```

```
 53                      Actions.run(new Runnable() {
 54                          public void run() { // Hooray!!!
 55                              spellingGameScreen.winConfetti(currentWord.getWordId());
 56                          }
 57                      }),
 58                      Actions.delay(0.5f),
 59                      Actions.run(new Runnable() {
 60                          public void run() { // Once final correct letter is dropped
    , say word
 61                              spellingGameScreen.playWord(currentLanguage.fileName,
    currentWord);
 62                          }
 63                      }),
 64                      Actions.delay(2f),
 65                      Actions.run(new Runnable() {
 66                          public void run() { // Then play word SFX
 67                              spellingGameScreen.playSFX(currentWord.getWordId());
 68                          }
 69                      }),
 70                      Actions.delay(1f),
 71                      Actions.run(new Runnable() {
 72                          public void run() {
 73                              spellingGameScreen.playSFX("applause");
 74                              changeToNextWord();
 75                          }
 76                      })
 77              ));
 78          } else { // Word isn't correct (yet): Play letter after every drop
 79              spellingGameScreen.playLetter(currentLanguage.fileName, (Letter) actor);
 80              if (spellingGameScreen.spacesFull()) { // Spaces full & not correct
 81                  spellingGameScreen.playSFX("buzzer");
 82              }
 83          }
 84      }
 85
 86      private boolean wordIsCorrect() {
 87          return currentWord.getSpelling(currentLanguage).equalsIgnoreCase(
    spellingGameScreen.getWordInSpaces());
 88      }
 89
 90      private void recordWord() {
 91          ScreenManager.getSelectedStudent().addToCurrentHistory(currentWord.
    getSpelling(currentLanguage));
 92      }
 93
 94      /**
 95       * Change word until all 15 words have been either spelled or skipped over
 96       */
 97      public void changeToNextWord() {
 98          if (sessionWordList.size() > 1) {
 99              sessionWordList.remove(currentWord);
100              currentWord = getNextWord();
101              // Set the amount of spaces for this word and replace the hint popup.
```

```
102             spellingGameScreen.setPictureAndSpaceLength(currentWord.getWordId(),
    currentWord.getSpaceLength(currentLanguage));
103             spellingGameScreen.hintPopup.clearActions();
104             spellingGameScreen.hintPopup.getColor().a = 0;
105             spellingGameScreen.hintPopup.setText(currentWord.getSpelling(
    currentLanguage));
106         } else {
107             gameComplete();
108         }
109     }
110
111     private Word getNextWord() {
112         if (sessionWordList.size() > 1) {
113             return sessionWordList.get(random.nextInt(sessionWordList.size() - 1));
114         } else {
115             return sessionWordList.get(0);
116         }
117     }
118
119     private void gameComplete() {
120         InputEvent event = new InputEvent();
121         event.setType(InputEvent.Type.touchDown);
122         spellingGameScreen.backButton.fire(event);
123         event.setType(InputEvent.Type.touchUp);
124         spellingGameScreen.backButton.fire(event);
125     }
126 }
127
```