Table of Contents

[Appendix C – Source Code Listing 1](#_Toc480364191)

[src.com.mygdx.game.audio 1](#_Toc480364192)

[src.com.mygdx.game.battle; 6](#_Toc480364193)

[src.com com.mygdx.game.components 12](#_Toc480364194)

[src.com.mygdx.game.gui; 49](#_Toc480364195)

[src.com.mygdx.game.inventory 102](#_Toc480364196)

[src.com.mygdx.game.japanese 110](#_Toc480364197)

[src.com.mygdx.game.maps 162](#_Toc480364198)

[src.com.mygdx.game.profile 192](#_Toc480364199)

[src.com.mygdx.game.screens 196](#_Toc480364200)

[src.com.mygdx.game.tools 219](#_Toc480364201)

[src.com.mygdx.game.desktop 237](#_Toc480364202)

[json\_scripts 238](#_Toc480364203)

[Android\_Code 280](#_Toc480364204)

# 

# Appendix C – Source Code Listing

## src.com.mygdx.game.audio

**package** com.mygdx.game.audio;   
  
**import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.audio.Music;  
**import** com.badlogic.gdx.audio.Sound;  
**import** com.mygdx.game.tools.Utility;  
  
**import** java.util.Hashtable;  
  
**public class** AudioManager **implements** AudioObserver {  
 **private static final** String ***TAG*** = AudioManager.**class**.getSimpleName();  
  
 **private static** AudioManager *\_instance* = **null**;  
  
 **private** Hashtable<String, Music> **queuedMusic**;  
 **private** Hashtable<String, Sound> **\_queuedSounds**;  
  
 **private** AudioManager(){  
 **queuedMusic** = **new** Hashtable<String, Music>();  
 **\_queuedSounds** = **new** Hashtable<String, Sound>();  
  
 }  
  
 **public static** AudioManager getInstance() {  
 **if** (*\_instance* == **null**) {  
 *\_instance* = **new** AudioManager();  
 }  
  
 **return** *\_instance*;  
 }  
  
 @Override  
 **public void** onNotify(AudioCommand command, AudioTypeEvent event) {  
 **switch**(command){  
 **case *MUSIC\_LOAD***:  
 Utility.*loadMusicAsset*(event.getValue());  
 **break**;  
 **case *MUSIC\_PLAY\_ONCE***:  
 playMusic(**false**, event.getValue());  
 **break**;  
 **case *MUSIC\_PLAY\_LOOP***:  
 playMusic(**true**, event.getValue());  
 **break**;  
 **case *MUSIC\_STOP***:  
 Music music = **queuedMusic**.get(event.getValue());  
 **if**( music != **null** ){  
 music.stop();  
 }  
 **break**;  
 **case *MUSIC\_STOP\_ALL***:  
 **for**( Music musicStop: **queuedMusic**.values() ){  
 musicStop.stop();  
 }  
 **break**;  
 **case *SOUND\_LOAD***:  
 Utility.*loadSoundAsset*(event.getValue());  
 **break**;  
 **case *SOUND\_PLAY\_LOOP***:  
 playSound(**true**, event.getValue());  
 **break**;  
 **case *SOUND\_PLAY\_ONCE***:  
 playSound(**false**, event.getValue());  
 **break**;  
 **case *SOUND\_STOP***:  
 Sound sound = **\_queuedSounds**.get(event.getValue());  
 **if**( sound != **null** ){  
 sound.stop();  
 }  
 **break**;  
 **default**:  
 **break**;  
 }  
 }  
  
 **private** Music playMusic(**boolean** isLooping, String fullFilePath){  
 Music music = **queuedMusic**.get(fullFilePath);  
 **if**( music != **null** ){  
 music.setLooping(isLooping);  
 music.play();  
 }**else if**(Utility.*isAssetLoaded*(fullFilePath)){  
 music = Utility.*getMusicAsset*(fullFilePath);  
 music.setLooping(isLooping);  
 music.play();  
 **queuedMusic**.put(fullFilePath, music);  
 }**else**{  
 Gdx.*app*.debug(***TAG***, **"Music not loaded"**);  
 **return null**;  
 }  
 **return** music;  
 }  
  
 **private** Sound playSound(**boolean** isLooping, String fullFilePath){  
 Sound sound = **\_queuedSounds**.get(fullFilePath);  
 **if**( sound != **null** ){  
 **long** soundId = sound.play();  
 sound.setLooping(soundId, isLooping);  
 }**else if**( Utility.*isAssetLoaded*(fullFilePath) ) {  
 sound = Utility.*getSoundAsset*(fullFilePath);  
 **long** soundId = sound.play();  
 sound.setLooping(soundId, isLooping);  
 **\_queuedSounds**.put(fullFilePath, sound);  
 }**else**{  
 Gdx.*app*.debug(***TAG***, **"Sound not loaded"**);  
 **return null**;  
 }  
 **return** sound;  
 }  
  
 **public void** dispose(){  
 **for**(Music music: **queuedMusic**.values()){  
 music.dispose();  
 }  
  
 **for**(Sound sound: **\_queuedSounds**.values()){  
 sound.dispose();  
 }  
 }  
  
}

**package** com.mygdx.game.audio;  
  
**public interface** AudioObserver {  
 **public static enum** AudioTypeEvent{  
 ***MUSIC\_TITLE***(**"audio/main\_music.ogg"**),  
 ***MUSIC\_TOPWORLD***(**"audio/topworld\_music.ogg"**),  
 ***MUSIC\_TOWN***(**"audio/town\_music.mp3"**),  
 ***MUSIC\_BATTLE***(**"audio/random\_battle.mp3"**),  
 ***MUSIC\_FOREST***(**"audio/forest.mp3"**),  
 ***MUSIC\_CAVE***(**"audio/cave\_music.mp3"**),  
 ***MUSIC\_COAST***(**"audio/coast\_music.mp3"**),  
 ***MUSIC\_ICE\_FOREST***(**"audio/ice\_forest\_music.mp3"**),  
 ***MUSIC\_DESERT\_TEMPLE***(**"audio/desert\_temple\_music.mp3"**),  
 ***MUSIC\_LAVA***(**"audio/lava\_music.mp3"**),  
 ***SOUND\_PLAYER\_PAIN***(**"audio/player\_hit.wav"**),  
 ***SOUND\_EATING***(**"audio/eating\_sound.wav"**),  
 ***SOUND\_DRINKING***(**"audio/liquid\_drinking.wav"**),  
 ***SOUND\_SUCCESS***(**"audio/success.wav"**),  
 ***SOUND\_LVL\_UP***(**"audio/level\_up.wav"**),  
 ***SOUND\_LVL\_DOWN***(**"audio/level\_down.wav"**),  
 ***NONE***(**""**);  
  
 **private** String **audioFullFilePath**;  
  
 AudioTypeEvent(String audioFullFilePath){  
 **this**.**audioFullFilePath** = audioFullFilePath;  
 }  
  
 **public** String getValue(){  
 **return audioFullFilePath**;  
 }  
 }  
  
 **public static enum** AudioCommand {  
 ***MUSIC\_LOAD***,  
 ***MUSIC\_PLAY\_ONCE***,  
 ***MUSIC\_PLAY\_LOOP***,  
 ***MUSIC\_STOP***,  
 ***MUSIC\_STOP\_ALL***,  
 ***SOUND\_LOAD***,  
 ***SOUND\_PLAY\_ONCE***,  
 ***SOUND\_PLAY\_LOOP***,  
 ***SOUND\_STOP*** }  
  
 **void** onNotify(AudioCommand command, AudioTypeEvent event);  
}

**package** com.mygdx.game.audio;  
  
**public interface** AudioSubject {  
 **public void** addObserver(AudioObserver audioObserver);  
 **public void** removeObserver(AudioObserver audioObserver);  
 **public void** removeAllObservers();  
 **public void** notify(**final** AudioObserver.AudioCommand command, AudioObserver.AudioTypeEvent event);  
}

## src.com.mygdx.game.battle;

**package** com.mygdx.game.battle;  
  
**import** com.mygdx.game.tools.Entity;  
  
**public interface** BattleObserver {  
 **public static enum** BattleEvent{  
 ***KANJI\_ADDED***,  
 ***HIRAGANA\_ADDED***,  
 ***KATAKANA\_ADDED***,  
 ***LETTER\_ANSWERED\_CORRECTLY***,  
 ***LETTER\_ANSWERED\_INCORRECTLY***,  
 ***PLAYER\_HIT\_DAMAGE***,  
 ***PLAYER\_RUNNING***,  
 ***PLAYER\_HAS\_MOVED***,  
 ***NONE*** }  
  
 **void** onNotify(**final** String answeredLetter, BattleEvent event);  
}

**package** com.mygdx.game.battle;  
  
**import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.math.MathUtils;  
**import** com.mygdx.game.japanese.KanaLettersFactory;  
**import** com.mygdx.game.japanese.LetterLvlCounter;  
  
**public class** BattleState **extends** BattleSubject {  
  
 *//****TODO speak about this,* private static final** String ***TAG*** = BattleState.**class**.getSimpleName();  
  
  
 **private int currentZoneLevel** = 0;  
 **private final int chanceOfEncounter** = 65;  
 **private final int chanceOfEscape** = 85;  
 **private** String **currentLetter** = **""**;  
  
 **public void** setCurrentZoneLevel(**int** zoneLevel){  
 **currentZoneLevel** = zoneLevel;  
 }  
  
 **public int** getCurrentZoneLevel(){  
 **return currentZoneLevel**;  
 }  
  
 **public boolean** isOpponentReady(){  
 **if**( **currentZoneLevel** == 0 ) **return false**;  
 **int** randomVal = MathUtils.*random*(1,100);  
  
 **if**( **chanceOfEncounter** > randomVal ){  
 setCurrentOpponent();  
 **return true**;  
 }**else**{  
 **return false**;  
 }  
 }  
  
 *//****TODO speak about this,* public void** setCurrentOpponent(){  
 Gdx.*app*.debug(***TAG***, **" Entered BATTLE ZONE: "** + **currentZoneLevel**);  
 **int** randomVal;  
 String letterToAnswer;  
 **int** letterToAnswersLvl;  
 **int** cycleCounter;  
 **if**(!LetterLvlCounter.*isAllHiraganaMemorised*()){  
 cycleCounter = 0;  
 randomVal= MathUtils.*random*(0,106);  
 letterToAnswer = KanaLettersFactory.*getInstance*().getKanaLettersList().get(randomVal).getHiraganaEquivalent();  
 **if** (letterToAnswer == **null**) **return**;  
  
 letterToAnswersLvl = LetterLvlCounter.*getHiraganaLvlTable*().get(letterToAnswer);  
 **while**(letterToAnswersLvl >= 3 || **currentLetter**.equalsIgnoreCase(letterToAnswer)){  
 **if**(cycleCounter >= 106){**break**;} **else** {cycleCounter++;}  
  
 randomVal= MathUtils.*random*(0,106);  
 letterToAnswer = KanaLettersFactory.*getInstance*().getKanaLettersList().get(randomVal).getHiraganaEquivalent();  
 letterToAnswersLvl = LetterLvlCounter.*getHiraganaLvlTable*().get(letterToAnswer);  
 }  
  
 **this**.**currentLetter** = letterToAnswer;  
 notify(letterToAnswer, BattleObserver.BattleEvent.***HIRAGANA\_ADDED***);  
 }  
 **else if**(!LetterLvlCounter.*isAllKatakanaMemorised*()){  
 cycleCounter = 0;  
 randomVal = MathUtils.*random*(0,106);  
 letterToAnswer = KanaLettersFactory.*getInstance*().getKanaLettersList().get(randomVal).getKatakanaEquivalent();  
 **if** (letterToAnswer == **null**) **return**;  
  
 letterToAnswersLvl = LetterLvlCounter.*getKatakanaLvlTable*().get(letterToAnswer);  
 **while**(letterToAnswersLvl >= 3 || **currentLetter**.equalsIgnoreCase(letterToAnswer)){  
 **if**(cycleCounter >= 106){**break**;} **else** {cycleCounter++;}  
 randomVal= MathUtils.*random*(0,106);  
 letterToAnswer = KanaLettersFactory.*getInstance*().getKanaLettersList().get(randomVal).getKatakanaEquivalent();  
 letterToAnswersLvl = LetterLvlCounter.*getKatakanaLvlTable*().get(letterToAnswer);  
 }  
 **this**.**currentLetter** = letterToAnswer;  
 notify(letterToAnswer, BattleObserver.BattleEvent.***KATAKANA\_ADDED***);  
 } **else** {  
 cycleCounter = 0;  
 letterToAnswer = EncounterFactory.*getInstance*().getRandomLetter(**currentZoneLevel**);  
 **if** (letterToAnswer == **null**) **return**;  
  
 letterToAnswersLvl = LetterLvlCounter.*getKanjiLvlTable*().get(letterToAnswer);  
 **while**(letterToAnswersLvl >= 5 || **currentLetter**.equalsIgnoreCase(letterToAnswer)){  
 **if**(cycleCounter >= 5){**break**;} **else** {cycleCounter++;}  
 letterToAnswer = EncounterFactory.*getInstance*().getRandomLetter(**currentZoneLevel**);  
 letterToAnswersLvl = LetterLvlCounter.*getKanjiLvlTable*().get(letterToAnswer);  
 }  
 **this**.**currentLetter** = letterToAnswer;  
 notify(letterToAnswer, BattleObserver.BattleEvent.***KANJI\_ADDED***);  
 }  
 }  
  
 **public void** playerRuns(){  
 **int** randomVal = MathUtils.*random*(1,100);  
 **if**( **chanceOfEscape** > randomVal ) {  
 notify(**currentLetter**, BattleObserver.BattleEvent.***PLAYER\_RUNNING***);  
 }**else**{  
 playerHit();  
 **return**;  
 }  
 }  
  
 **public void** playerHit(){  
 **if**( **currentLetter** == **null** ){  
 **return**;  
 }  
  
 notify(**currentLetter**, BattleObserver.BattleEvent.***PLAYER\_HIT\_DAMAGE***);  
  
 Gdx.*app*.debug(***TAG***, **"PLayer lost health"**);  
 }  
  
 **public void** answeredCorrectly(String answeredLetter){  
 notify(answeredLetter, BattleObserver.BattleEvent.***LETTER\_ANSWERED\_CORRECTLY***);  
 }  
  
  
 **public void** answeredIncorrectly(String answeredLetter){  
 **if**( **currentLetter** == **null** ){  
 **return**;  
 }  
  
 notify(**currentLetter**, BattleObserver.BattleEvent.***LETTER\_ANSWERED\_INCORRECTLY***);  
  
 Gdx.*app*.debug(***TAG***, **"PLayer lost health"**);  
 }  
}

**package** com.mygdx.game.battle;  
  
**import** com.badlogic.gdx.math.MathUtils;  
**import** com.badlogic.gdx.utils.Array;  
  
**import** java.util.Hashtable;  
  
**public class** EncounterFactory {  
  
 **private static** EncounterFactory *\_instance* = **null**;  
 **private** Hashtable<String,Array<String> > **encounterZones**;  
  
 **public static** EncounterFactory getInstance() {  
 **if** (*\_instance* == **null**) {  
 *\_instance* = **new** EncounterFactory();  
 }  
  
 **return** *\_instance*;  
 }  
  
 **private** EncounterFactory(){  
 **encounterZones** = EncounterZone.*getMonsterZones*(**"json\_scripts/encounter\_zones.json"**);  
 }  
  
  
 **public** String getRandomLetter(**int** monsterZoneID){  
 Array<String> letters = **encounterZones**.get(String.*valueOf*(monsterZoneID));  
 **int** size = letters.**size**;  
 **if**( size == 0 ){  
 **return null**;  
 }  
 **int** randomIndex = MathUtils.*random*(size - 1);  
  
 **return** letters.get(randomIndex);  
 }  
  
  
}

**package** com.mygdx.game.battle;  
  
**import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.utils.Array;  
**import** com.badlogic.gdx.utils.Json;  
**import** com.badlogic.gdx.utils.JsonValue;  
  
**import** java.util.ArrayList;  
**import** java.util.Hashtable;  
  
**public class** EncounterZone {  
 **private** String **zoneID**;  
 **private** String **monsters**;  
  
 **public** String getZoneID() {  
 **return zoneID**;  
 }  
  
 **public void** setZoneID(String zoneID) {  
 **this**.**zoneID** = zoneID;  
 }  
  
 **public** String getMonsters() {  
 **return monsters**;  
 }  
  
 **public void** setMonsters(String monsters) {  
 **this**.**monsters** = monsters;  
 }  
  
 **static public** Hashtable<String, Array<String>> getMonsterZones(String configFilePath){  
 Json json = **new** Json();  
 Hashtable<String, Array<String>> monsterZones = **new** Hashtable<String, Array<String>>();  
  
 *//****TODO talk about this,*** ArrayList<JsonValue> list = json.fromJson(ArrayList.**class**, Gdx.*files*.internal(configFilePath));  
  
 String temp;  
 String split;  
 **int** indexOfFullStop;  
 Array<String> monster;  
  
 **for** (JsonValue jsonVal : list) {  
 EncounterZone zone = json.readValue(EncounterZone.**class**, jsonVal);  
  
 temp = zone.getMonsters();  
 monster = **new** Array<>();  
  
 **for** (**int** j = 0; j < temp.length(); j++) {  
 **if** (temp.charAt(j) == **'.'**) {  
 **try** {  
 indexOfFullStop = temp.indexOf(**'.'**, j + 1);  
 split = temp.substring(j + 1, indexOfFullStop);  
 monster.add(split);  
 } **catch** (Exception e) {  
 }  
  
 }  
 }  
  
 monsterZones.put(zone.getZoneID(), monster);  
 }  
  
 **return** monsterZones;  
 }  
}

## src.com com.mygdx.game.components

**package** com.mygdx.game.components;  
  
  
**public interface** Component {  
  
 **public static final** String ***MESSAGE\_TOKEN*** = **":::::"**;  
  
 **public static enum** MESSAGE{  
 ***CURRENT\_POSITION***,  
 ***INIT\_START\_POSITION***,  
 ***CURRENT\_DIRECTION***,  
 ***CURRENT\_STATE***,  
 ***COLLISION\_WITH\_MAP***,  
 ***COLLISION\_WITH\_ENTITY***,  
 ***LOAD\_ANIMATIONS***,  
 ***INIT\_DIRECTION***,  
 ***INIT\_STATE***,  
 ***INIT\_SELECT\_ENTITY***,  
 ***ENTITY\_SELECTED***,  
 ***ENTITY\_DESELECTED*** }  
  
 **void** dispose();  
 **void** receiveMessage(String message);  
}

**package** com.mygdx.game.components;  
  
**public interface** ComponentObserver {  
 **public static enum** ComponentEvent {  
 ***ENEMY\_SPAWN\_LOCATION\_CHANGED***,  
 ***PLAYER\_HAS\_MOVED*** }  
  
 **void** onNotify(**final** String value, ComponentEvent event);  
}

**package** com.mygdx.game.components;  
  
**import** com.badlogic.gdx.utils.Array;  
  
**public class** ComponentSubject {  
 **private** Array<ComponentObserver> **observers**;  
  
 **public** ComponentSubject(){  
 **observers** = **new** Array<ComponentObserver>();  
 }  
  
 **public void** addObserver(ComponentObserver conversationObserver){  
 **observers**.add(conversationObserver);  
 }  
  
 **public void** removeObserver(ComponentObserver conversationObserver){  
 **observers**.removeValue(conversationObserver, **true**);  
 }  
  
 **public void** removeAllObservers(){  
 **for**(ComponentObserver observer: **observers**){  
 **observers**.removeValue(observer, **true**);  
 }  
 }  
  
 **protected void** notify(**final** String value, ComponentObserver.ComponentEvent event){  
 **for**(ComponentObserver observer: **observers**){  
 observer.onNotify(value, event);  
 }  
 }  
}

**package** com.mygdx.game.components;  
  
**import** com.badlogic.gdx.graphics.Texture;  
**import** com.badlogic.gdx.graphics.g2d.Animation;  
**import** com.badlogic.gdx.graphics.g2d.Batch;  
**import** com.badlogic.gdx.graphics.g2d.TextureRegion;  
**import** com.badlogic.gdx.graphics.glutils.ShapeRenderer;  
**import** com.badlogic.gdx.math.GridPoint2;  
**import** com.badlogic.gdx.math.Vector2;  
**import** com.badlogic.gdx.utils.Array;  
**import** com.badlogic.gdx.utils.Json;  
**import** com.mygdx.game.tools.Entity;  
**import** com.mygdx.game.tools.Utility;  
**import** com.mygdx.game.maps.MapManager;  
  
**import** java.util.Hashtable;  
  
**public abstract class** GraphicsComponent **extends** ComponentSubject **implements** Component {  
 **protected** TextureRegion **currentFrame** = **null**;  
 **protected float frameTime** = 0f;  
 **protected** Entity.State **currentState**;  
 **protected** Entity.Direction **currentDirection**;  
 **protected** Json **json**;  
 **protected** Hashtable<Entity.AnimationType, Animation> **animations**;  
 **protected** ShapeRenderer **shapeRenderer**;  
  
 **public** Vector2 **currentPosition**;  
  
 **protected** GraphicsComponent(){  
 **currentPosition** = **new** Vector2(0,0);  
 **currentState** = Entity.State.***WALKING***;  
 **currentDirection** = Entity.Direction.***DOWN***;  
 **json** = **new** Json();  
 **animations** = **new** Hashtable<Entity.AnimationType, Animation>();  
 **shapeRenderer** = **new** ShapeRenderer();  
 }  
  
 **public abstract void** update(Entity entity, MapManager mapManager, Batch batch, **float** delta);  
  
 **protected void** updateAnimations(**float** delta){  
 **frameTime** = (**frameTime** + delta)%5; *//Want to avoid overflow  
  
 //Look into the appropriate variable when changing position* **switch** (**currentDirection**) {  
 **case *DOWN***:  
 **if** (**currentState** == Entity.State.***WALKING***) {  
 Animation animation = **animations**.get(Entity.AnimationType.***WALK\_DOWN***);  
 **if**( animation == **null** ) **return**;  
 **currentFrame** = animation.getKeyFrame(**frameTime**);  
 } **else if**(**currentState** == Entity.State.***IDLE***) {  
 Animation animation = **animations**.get(Entity.AnimationType.***WALK\_DOWN***);  
 **if**( animation == **null** ) **return**;  
 **currentFrame** = animation.getKeyFrames()[0];  
 } **else if**(**currentState** == Entity.State.***IMMOBILE***) {  
 Animation animation = **animations**.get(Entity.AnimationType.***IMMOBILE***);  
 **if**( animation == **null** ) **return**;  
 **currentFrame** = animation.getKeyFrame(**frameTime**);  
 }  
 **break**;  
 **case *DOWN\_LEFT***:  
 **if** (**currentState** == Entity.State.***WALKING***) {  
 Animation animation = **animations**.get(Entity.AnimationType.***WALK\_DOWN\_LEFT***);  
 **if**( animation == **null** ) **return**;  
 **currentFrame** = animation.getKeyFrame(**frameTime**);  
 } **else if**(**currentState** == Entity.State.***IDLE***) {  
 Animation animation = **animations**.get(Entity.AnimationType.***WALK\_DOWN\_LEFT***);  
 **if**( animation == **null** ) **return**;  
 **currentFrame** = animation.getKeyFrames()[0];  
 } **else if**(**currentState** == Entity.State.***IMMOBILE***) {  
 Animation animation = **animations**.get(Entity.AnimationType.***IMMOBILE***);  
 **if**( animation == **null** ) **return**;  
 **currentFrame** = animation.getKeyFrame(**frameTime**);  
 }  
 **break**;  
 **case *DOWN\_RIGHT***:  
 **if** (**currentState** == Entity.State.***WALKING***) {  
 Animation animation = **animations**.get(Entity.AnimationType.***WALK\_DOWN\_RIGHT***);  
 **if**( animation == **null** ) **return**;  
 **currentFrame** = animation.getKeyFrame(**frameTime**);  
 } **else if**(**currentState** == Entity.State.***IDLE***) {  
 Animation animation = **animations**.get(Entity.AnimationType.***WALK\_DOWN\_RIGHT***);  
 **if**( animation == **null** ) **return**;  
 **currentFrame** = animation.getKeyFrames()[0];  
 } **else if**(**currentState** == Entity.State.***IMMOBILE***) {  
 Animation animation = **animations**.get(Entity.AnimationType.***IMMOBILE***);  
 **if**( animation == **null** ) **return**;  
 **currentFrame** = animation.getKeyFrame(**frameTime**);  
 }  
 **break**;  
 **case *LEFT***:  
 **if** (**currentState** == Entity.State.***WALKING***) {  
 Animation animation = **animations**.get(Entity.AnimationType.***WALK\_LEFT***);  
 **if**( animation == **null** ) **return**;  
 **currentFrame** = animation.getKeyFrame(**frameTime**);  
 } **else if**(**currentState** == Entity.State.***IDLE***) {  
 Animation animation = **animations**.get(Entity.AnimationType.***WALK\_LEFT***);  
 **if**( animation == **null** ) **return**;  
 **currentFrame** = animation.getKeyFrames()[0];  
 } **else if**(**currentState** == Entity.State.***IMMOBILE***) {  
 Animation animation = **animations**.get(Entity.AnimationType.***IMMOBILE***);  
 **if**( animation == **null** ) **return**;  
 **currentFrame** = animation.getKeyFrame(**frameTime**);  
 }  
 **break**;  
 **case *UP***:  
 **if** (**currentState** == Entity.State.***WALKING***) {  
 Animation animation = **animations**.get(Entity.AnimationType.***WALK\_UP***);  
 **if**( animation == **null** ) **return**;  
 **currentFrame** = animation.getKeyFrame(**frameTime**);  
 } **else if**(**currentState** == Entity.State.***IDLE***) {  
 Animation animation = **animations**.get(Entity.AnimationType.***WALK\_UP***);  
 **if**( animation == **null** ) **return**;  
 **currentFrame** = animation.getKeyFrames()[0];  
 } **else if**(**currentState** == Entity.State.***IMMOBILE***) {  
 Animation animation = **animations**.get(Entity.AnimationType.***IMMOBILE***);  
 **if**( animation == **null** ) **return**;  
 **currentFrame** = animation.getKeyFrame(**frameTime**);  
 }  
 **break**;  
 **case *UP\_LEFT***:  
 **if** (**currentState** == Entity.State.***WALKING***) {  
 Animation animation = **animations**.get(Entity.AnimationType.***WALK\_UP\_LEFT***);  
 **if**( animation == **null** ) **return**;  
 **currentFrame** = animation.getKeyFrame(**frameTime**);  
 } **else if**(**currentState** == Entity.State.***IDLE***) {  
 Animation animation = **animations**.get(Entity.AnimationType.***WALK\_UP\_LEFT***);  
 **if**( animation == **null** ) **return**;  
 **currentFrame** = animation.getKeyFrames()[0];  
 } **else if**(**currentState** == Entity.State.***IMMOBILE***) {  
 Animation animation = **animations**.get(Entity.AnimationType.***IMMOBILE***);  
 **if**( animation == **null** ) **return**;  
 **currentFrame** = animation.getKeyFrame(**frameTime**);  
 }  
 **break**;  
 **case *UP\_RIGHT***:  
 **if** (**currentState** == Entity.State.***WALKING***) {  
 Animation animation = **animations**.get(Entity.AnimationType.***WALK\_UP\_RIGHT***);  
 **if**( animation == **null** ) **return**;  
 **currentFrame** = animation.getKeyFrame(**frameTime**);  
 } **else if**(**currentState** == Entity.State.***IDLE***) {  
 Animation animation = **animations**.get(Entity.AnimationType.***WALK\_UP\_RIGHT***);  
 **if**( animation == **null** ) **return**;  
 **currentFrame** = animation.getKeyFrames()[0];  
 } **else if**(**currentState** == Entity.State.***IMMOBILE***) {  
 Animation animation = **animations**.get(Entity.AnimationType.***IMMOBILE***);  
 **if**( animation == **null** ) **return**;  
 **currentFrame** = animation.getKeyFrame(**frameTime**);  
 }  
 **break**;  
 **case *RIGHT***:  
 **if** (**currentState** == Entity.State.***WALKING***) {  
 Animation animation = **animations**.get(Entity.AnimationType.***WALK\_RIGHT***);  
 **if**( animation == **null** ) **return**;  
 **currentFrame** = animation.getKeyFrame(**frameTime**);  
 } **else if**(**currentState** == Entity.State.***IDLE***) {  
 Animation animation = **animations**.get(Entity.AnimationType.***WALK\_RIGHT***);  
 **if**( animation == **null** ) **return**;  
 **currentFrame** = animation.getKeyFrames()[0];  
 } **else if**(**currentState** == Entity.State.***IMMOBILE***) {  
 Animation animation = **animations**.get(Entity.AnimationType.***IMMOBILE***);  
 **if**( animation == **null** ) **return**;  
 **currentFrame** = animation.getKeyFrame(**frameTime**);  
 }  
 **break**;  
 **default**:  
 **break**;  
 }  
 }  
  
 *//Specific to two frame animations where each frame is stored in a separate texture* **protected** Animation loadAnimation(String firstTexture, String secondTexture, Array<GridPoint2> points, **float** frameDuration){  
 Utility.*loadTextureAsset*(firstTexture);  
 Texture texture1 = Utility.*getTextureAsset*(firstTexture);  
  
 Utility.*loadTextureAsset*(secondTexture);  
 Texture texture2 = Utility.*getTextureAsset*(secondTexture);  
  
 TextureRegion[][] texture1Frames = TextureRegion.*split*(texture1, Entity.*FRAME\_WIDTH*, Entity.*FRAME\_HEIGHT*);  
 TextureRegion[][] texture2Frames = TextureRegion.*split*(texture2, Entity.*FRAME\_WIDTH*, Entity.*FRAME\_HEIGHT*);  
  
 Array<TextureRegion> animationKeyFrames = **new** Array<TextureRegion>(2);  
  
 GridPoint2 point = points.first();  
  
 animationKeyFrames.add(texture1Frames[point.**x**][point.**y**]);  
 animationKeyFrames.add(texture2Frames[point.**x**][point.**y**]);  
  
 **return new** Animation(frameDuration, animationKeyFrames, Animation.PlayMode.***LOOP***);  
 }  
  
 **protected** Animation loadAnimation(String textureName, Array<GridPoint2> points, **float** frameDuration){  
 Utility.*loadTextureAsset*(textureName);  
 Texture texture = Utility.*getTextureAsset*(textureName);  
  
 TextureRegion[][] textureFrames = TextureRegion.*split*(texture, Entity.*FRAME\_WIDTH*, Entity.*FRAME\_HEIGHT*);  
  
 Array<TextureRegion> animationKeyFrames = **new** Array<TextureRegion>(points.**size**);  
  
 **for**( GridPoint2 point : points){  
 animationKeyFrames.add(textureFrames[point.**x**][point.**y**]);  
 }  
  
 **return new** Animation(frameDuration, animationKeyFrames, Animation.PlayMode.***LOOP***);  
 }  
  
 **public** Animation getAnimation(Entity.AnimationType type){  
 **return animations**.get(type);  
 }  
  
}

**package** com.mygdx.game.components;  
  
**import** com.badlogic.gdx.InputProcessor;  
**import** com.badlogic.gdx.utils.Json;  
**import** com.mygdx.game.tools.Entity;  
  
**import** java.util.HashMap;  
**import** java.util.Map;  
  
**public abstract class** InputComponent **extends** ComponentSubject **implements** Component, InputProcessor {  
  
 **protected** Entity.Direction **currentDirection** = **null**;  
 **protected** Entity.State **currentState** = **null**;  
 **protected** Json **json**;  
  
 **protected enum** Keys {  
 ***LEFT***, ***RIGHT***, ***UP***, ***DOWN***, ***QUIT***, ***PAUSE***,  
 ***UP\_RIGHT***, ***UP\_LEFT***, ***DOWN\_RIGHT***, ***DOWN\_LEFT*** }  
  
 **protected enum** Mouse {  
 ***SELECT***, ***DOACTION*** }  
  
 **protected static** Map<Keys, Boolean> *keys* = **new** HashMap<Keys, Boolean>();  
 **protected static** Map<Mouse, Boolean> *mouseButtons* = **new** HashMap<Mouse, Boolean>();  
  
 *//initialize the hashmap for inputs* **static** {  
 *keys*.put(Keys.***LEFT***, **false**);  
 *keys*.put(Keys.***RIGHT***, **false**);  
 *keys*.put(Keys.***UP***, **false**);  
 *keys*.put(Keys.***DOWN***, **false**);  
 *keys*.put(Keys.***UP\_LEFT***, **false**);  
 *keys*.put(Keys.***UP\_RIGHT***, **false**);  
 *keys*.put(Keys.***DOWN\_LEFT***, **false**);  
 *keys*.put(Keys.***DOWN\_RIGHT***, **false**);  
 *keys*.put(Keys.***QUIT***, **false**);  
 *keys*.put(Keys.***PAUSE***, **false**);  
 };  
  
 **static** {  
 *mouseButtons*.put(Mouse.***SELECT***, **false**);  
 *mouseButtons*.put(Mouse.***DOACTION***, **false**);  
 };  
  
 InputComponent(){  
 **json** = **new** Json();  
 }  
  
 **public abstract void** update(Entity entity, **float** delta);  
  
}

**package** com.mygdx.game.components;  
  
**import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.graphics.Camera;  
**import** com.badlogic.gdx.graphics.GL20;  
**import** com.badlogic.gdx.graphics.g2d.Animation;  
**import** com.badlogic.gdx.graphics.g2d.Batch;  
**import** com.badlogic.gdx.graphics.glutils.ShapeRenderer;  
**import** com.badlogic.gdx.math.GridPoint2;  
**import** com.badlogic.gdx.math.Rectangle;  
**import** com.badlogic.gdx.math.Vector2;  
**import** com.badlogic.gdx.utils.Array;  
**import** com.mygdx.game.tools.Entity;  
**import** com.mygdx.game.tools.EntityConfig;  
**import** com.mygdx.game.tools.EntityConfig.AnimationConfig;  
**import** com.mygdx.game.maps.Map;  
**import** com.mygdx.game.maps.MapManager;  
  
**public class** NPCGraphicsComponent **extends** GraphicsComponent {  
  
 **private static final** String ***TAG*** = NPCGraphicsComponent.**class**.getSimpleName();  
  
 **private boolean isSelected** = **false**;  
  
 **public** NPCGraphicsComponent(){  
 }  
  
 @Override  
 **public void** receiveMessage(String message) {  
 *//Gdx.app.debug(TAG, "Got message " + message);* String[] string = message.split(***MESSAGE\_TOKEN***);  
  
 **if**( string.**length** == 0 ) **return**;  
  
 **if**( string.**length** == 1 ) {  
 **if** (string[0].equalsIgnoreCase(MESSAGE.***ENTITY\_SELECTED***.toString())) {  
 **isSelected** = **true**;  
 }**else if** (string[0].equalsIgnoreCase(MESSAGE.***ENTITY\_DESELECTED***.toString())) {  
 **isSelected** = **false**;  
 }  
 }  
  
 *//Specifically for messages with 1 object payload* **if**( string.**length** == 2 ) {  
 **if** (string[0].equalsIgnoreCase(MESSAGE.***CURRENT\_POSITION***.toString())) {  
 **currentPosition** = **json**.fromJson(Vector2.**class**, string[1]);  
 } **else if** (string[0].equalsIgnoreCase(MESSAGE.***INIT\_START\_POSITION***.toString())) {  
 **currentPosition** = **json**.fromJson(Vector2.**class**, string[1]);  
 } **else if** (string[0].equalsIgnoreCase(MESSAGE.***CURRENT\_STATE***.toString())) {  
 **currentState** = **json**.fromJson(Entity.State.**class**, string[1]);  
 } **else if** (string[0].equalsIgnoreCase(MESSAGE.***CURRENT\_DIRECTION***.toString())) {  
 **currentDirection** = **json**.fromJson(Entity.Direction.**class**, string[1]);  
 }**else if** (string[0].equalsIgnoreCase(MESSAGE.***LOAD\_ANIMATIONS***.toString())) {  
 EntityConfig entityConfig = **json**.fromJson(EntityConfig.**class**, string[1]);  
 Array<EntityConfig.AnimationConfig> animationConfigs = entityConfig.getAnimationConfig();  
  
 **for**( AnimationConfig animationConfig : animationConfigs ){  
 Array<String> textureNames = animationConfig.getTexturePaths();  
 Array<GridPoint2> points = animationConfig.getGridPoints();  
 Entity.AnimationType animationType = animationConfig.getAnimationType();  
 **float** frameDuration = animationConfig.getFrameDuration();  
 Animation animation = **null**;  
  
 **if**( textureNames.**size** == 1) {  
 animation = loadAnimation(textureNames.get(0), points, frameDuration);  
 }**else if**( textureNames.**size** == 2){  
 animation = loadAnimation(textureNames.get(0), textureNames.get(1), points, frameDuration);  
 }  
  
 **animations**.put(animationType, animation);  
 }  
 }  
 }  
 }  
  
 @Override  
 **public void** update(Entity entity, MapManager mapMgr, Batch batch, **float** delta){  
 updateAnimations(delta);  
  
 **if**(**isSelected**){  
 drawSelected(entity, mapMgr);  
 }  
  
 batch.begin();  
 batch.draw(**currentFrame**, **currentPosition**.**x**, **currentPosition**.**y**, 1.2f, 2.2f);  
 batch.end();  
  
*// Used to graphically debug boundingboxes  
// Rectangle rect = tools.getCurrentBoundingBox();  
// Camera camera = mapMgr.getCamera();  
// shapeRenderer.setProjectionMatrix(camera.combined);  
// shapeRenderer.begin(ShapeRenderer.ShapeType.Filled);  
// shapeRenderer.setColor(Color.BLACK);  
// shapeRenderer.rect(rect.getX() \* Map.UNIT\_SCALE, rect.getY() \* Map.UNIT\_SCALE, rect.getWidth() \* Map.UNIT\_SCALE, rect.getHeight() \* Map.UNIT\_SCALE);  
// shapeRenderer.end();* }  
  
 **private void** drawSelected(Entity entity, MapManager mapMgr){  
 Gdx.*gl*.glEnable(GL20.***GL\_BLEND***);  
 Gdx.*gl*.glBlendFunc(GL20.***GL\_SRC\_ALPHA***, GL20.***GL\_ONE\_MINUS\_SRC\_ALPHA***);  
 Camera camera = mapMgr.getCamera();  
 Rectangle rect = entity.getCurrentBoundingBox();  
 **shapeRenderer**.setProjectionMatrix(camera.**combined**);  
 **shapeRenderer**.begin(ShapeRenderer.ShapeType.***Filled***);  
 **shapeRenderer**.setColor(0.0f, 1.0f, 1.0f, 0.5f);  
  
 **float** width = rect.getWidth() \* Map.***UNIT\_SCALE***\*2f;  
 **float** height = rect.getHeight() \* Map.***UNIT\_SCALE***/2f;  
 **float** x = rect.**x** \* Map.***UNIT\_SCALE*** - width/3;  
 **float** y = rect.**y** \* Map.***UNIT\_SCALE*** - height/1.5f;  
  
 **shapeRenderer**.ellipse(x,y,width,height);  
 **shapeRenderer**.end();  
 Gdx.*gl*.glDisable(GL20.***GL\_BLEND***);  
 }  
  
 @Override  
 **public void** dispose(){  
 }  
}

**package** com.mygdx.game.components;  
  
**import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.Input;  
**import** com.badlogic.gdx.InputProcessor;  
**import** com.badlogic.gdx.math.MathUtils;  
**import** com.mygdx.game.tools.Entity;  
  
**public class** NPCInputComponent **extends** InputComponent **implements** InputProcessor {  
  
 **private static final** String ***TAG*** = NPCInputComponent.**class**.getSimpleName();  
 **private float frameTime** = 0.0f;  
  
 **public** NPCInputComponent(){  
 **currentDirection** = Entity.Direction.*getRandomNext*();  
 **currentState** = Entity.State.***WALKING***;  
 }  
  
 @Override  
 **public void** receiveMessage(String message) {  
 String[] string = message.split(***MESSAGE\_TOKEN***);  
  
 **if**( string.**length** == 0 ) **return**;  
  
 *//Specifically for messages with 1 object payload* **if**( string.**length** == 1 ) {  
 **if** (string[0].equalsIgnoreCase(MESSAGE.***COLLISION\_WITH\_MAP***.toString())) {  
 **currentDirection** = Entity.Direction.*getRandomNext*();  
 }**else if** (string[0].equalsIgnoreCase(MESSAGE.***COLLISION\_WITH\_ENTITY***.toString())) {  
 **currentState** = Entity.State.***IDLE***;  
 *//currentDirection = currentDirection.getOpposite();* }  
 }  
  
 **if**( string.**length** == 2 ) {  
 **if** (string[0].equalsIgnoreCase(MESSAGE.***INIT\_STATE***.toString())) {  
 **currentState** = **json**.fromJson(Entity.State.**class**, string[1]);  
 }**else if** (string[0].equalsIgnoreCase(MESSAGE.***INIT\_DIRECTION***.toString())) {  
 **currentDirection** = **json**.fromJson(Entity.Direction.**class**, string[1]);  
 }  
 }  
  
 }  
  
 @Override  
 **public void** dispose(){  
  
 }  
  
 @Override  
 **public void** update(Entity entity, **float** delta){  
 **if**(*keys*.get(Keys.***QUIT***)) {  
 Gdx.*app*.exit();  
 }  
  
 *//If IMMOBILE, don't update anything* **if**( **currentState** == Entity.State.***IMMOBILE*** ) {  
 entity.sendMessage(MESSAGE.***CURRENT\_STATE***, **json**.toJson(Entity.State.***IMMOBILE***));  
 **return**;  
 }  
  
 **frameTime** += delta;  
  
 *//Change direction after so many seconds* **if**( **frameTime** > MathUtils.*random*(1,5) ){  
 **currentState** = Entity.State.*getRandomNext*();  
 **currentDirection** = Entity.Direction.*getRandomNext*();  
 **frameTime** = 0.0f;  
 }  
  
 **if**( **currentState** == Entity.State.***IDLE*** ){  
 entity.sendMessage(MESSAGE.***CURRENT\_STATE***, **json**.toJson(Entity.State.***IDLE***));  
 **return**;  
 }  
  
 **switch**(**currentDirection**) {  
 **case *LEFT***:  
 entity.sendMessage(MESSAGE.***CURRENT\_STATE***, **json**.toJson(Entity.State.***WALKING***));  
 entity.sendMessage(MESSAGE.***CURRENT\_DIRECTION***, **json**.toJson(Entity.Direction.***LEFT***));  
 **break**;  
 **case *RIGHT***:  
 entity.sendMessage(MESSAGE.***CURRENT\_STATE***, **json**.toJson(Entity.State.***WALKING***));  
 entity.sendMessage(MESSAGE.***CURRENT\_DIRECTION***, **json**.toJson(Entity.Direction.***RIGHT***));  
 **break**;  
 **case *UP***:  
 entity.sendMessage(MESSAGE.***CURRENT\_STATE***, **json**.toJson(Entity.State.***WALKING***));  
 entity.sendMessage(MESSAGE.***CURRENT\_DIRECTION***, **json**.toJson(Entity.Direction.***UP***));  
 **break**;  
 **case *DOWN***:  
 entity.sendMessage(MESSAGE.***CURRENT\_STATE***, **json**.toJson(Entity.State.***WALKING***));  
 entity.sendMessage(MESSAGE.***CURRENT\_DIRECTION***, **json**.toJson(Entity.Direction.***DOWN***));  
 **break**;  
 }  
 }  
  
 @Override  
 **public boolean** keyDown(**int** keycode) {  
 **if**( keycode == Input.Keys.***Q***){  
 *keys*.put(Keys.***QUIT***, **true**);  
 }  
 **return false**;  
 }  
  
 @Override  
 **public boolean** keyUp(**int** keycode) {  
 **return false**;  
 }  
  
 @Override  
 **public boolean** keyTyped(**char** character) {  
 **return false**;  
 }  
  
 @Override  
 **public boolean** touchDown(**int** screenX, **int** screenY, **int** pointer, **int** button) {  
 **return false**;  
 }  
  
 @Override  
 **public boolean** touchUp(**int** screenX, **int** screenY, **int** pointer, **int** button) {  
 **return false**;  
 }  
  
 @Override  
 **public boolean** touchDragged(**int** screenX, **int** screenY, **int** pointer) {  
 **return false**;  
 }  
  
 @Override  
 **public boolean** mouseMoved(**int** screenX, **int** screenY) {  
 **return false**;  
 }  
  
 @Override  
 **public boolean** scrolled(**int** amount) {  
 **return false**;  
 }  
}

**package** com.mygdx.game.components;  
  
**import** com.badlogic.gdx.math.Vector2;  
**import** com.mygdx.game.tools.Entity;  
**import** com.mygdx.game.maps.MapManager;  
  
**public class** NPCPhysicsComponent **extends** PhysicsComponent {  
  
 **private static final** String ***TAG*** = NPCPhysicsComponent.**class**.getSimpleName();  
  
 **private** Entity.State **state**;  
  
 **public** NPCPhysicsComponent(){  
 **boundingBoxLocation** = BoundingBoxLocation.***CENTER***;  
 initBoundingBox(0.6f, 0.4f);  
 **super**.**velocity**.set(3f,3f);  
 }  
  
 @Override  
 **public void** dispose(){  
 }  
  
 @Override  
 **public void** receiveMessage(String message) {  
 *//Gdx.app.debug(TAG, "Got message " + message);* String[] string = message.split(Component.***MESSAGE\_TOKEN***);  
  
 **if**( string.**length** == 0 ) **return**;  
  
 *//Specifically for messages with 1 object payload* **if**( string.**length** == 2 ) {  
 **if** (string[0].equalsIgnoreCase(MESSAGE.***INIT\_START\_POSITION***.toString())) {  
 **currentEntityPosition** = **json**.fromJson(Vector2.**class**, string[1]);  
 **nextEntityPosition**.set(**currentEntityPosition**.**x**, **currentEntityPosition**.**y**);  
 } **else if** (string[0].equalsIgnoreCase(MESSAGE.***CURRENT\_STATE***.toString())) {  
 **state** = **json**.fromJson(Entity.State.**class**, string[1]);  
 } **else if** (string[0].equalsIgnoreCase(MESSAGE.***CURRENT\_DIRECTION***.toString())) {  
 **currentDirection** = **json**.fromJson(Entity.Direction.**class**, string[1]);  
 }  
 }  
 }  
  
 @Override  
 **public void** update(Entity entity, MapManager mapMgr, **float** delta) {  
 updateBoundingBoxPosition(**nextEntityPosition**);  
  
 **if**( **state** == Entity.State.***IMMOBILE*** ) **return**;  
  
 **if** (!isCollisionWithMapLayer(entity, mapMgr) &&  
 !isCollisionWithMapEntities(entity, mapMgr) &&  
 **state** == Entity.State.***WALKING***){  
 setNextPositionToCurrent(entity);  
 } **else** {  
 updateBoundingBoxPosition(**currentEntityPosition**);  
 }  
 calculateNextPosition(delta);  
 }  
  
 @Override  
 **protected boolean** isCollisionWithMapEntities(Entity entity, MapManager mapMgr){  
 *//Test against player* **if**(isCollision(entity, mapMgr.getPlayer()) ) {  
 **return true**;  
 }  
  
 **if**(**super**.isCollisionWithMapEntities(entity, mapMgr) ){  
 **return true**;  
 }  
  
 **return false**;  
  
 }  
}

**package** com.mygdx.game.components;  
  
**import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.maps.MapLayer;  
**import** com.badlogic.gdx.maps.MapObject;  
**import** com.badlogic.gdx.maps.objects.RectangleMapObject;  
**import** com.badlogic.gdx.math.Rectangle;  
**import** com.badlogic.gdx.math.Vector2;  
**import** com.badlogic.gdx.utils.Array;  
**import** com.badlogic.gdx.utils.Json;  
**import** com.mygdx.game.tools.Entity;  
**import** com.mygdx.game.maps.Map;  
**import** com.mygdx.game.maps.MapManager;  
  
**public abstract class** PhysicsComponent **extends** ComponentSubject **implements** Component{  
  
 **private static final** String ***TAG*** = PhysicsComponent.**class**.getSimpleName();  
  
 **protected** Vector2 **nextEntityPosition**;  
 **protected** Vector2 **currentEntityPosition**;  
 **protected** Entity.Direction **currentDirection**;  
 **protected** Json **json**;  
 **protected** Vector2 **velocity**;  
  
 **public** Rectangle **boundingBox**;  
 **public** BoundingBoxLocation **boundingBoxLocation**;  
  
 **public static enum** BoundingBoxLocation{  
 ***BOTTOM\_LEFT***,  
 ***BOTTOM\_CENTER***,  
 ***CENTER***,  
 }  
  
 **protected** PhysicsComponent(){  
 **this**.**nextEntityPosition** = **new** Vector2(0,0);  
 **this**.**currentEntityPosition** = **new** Vector2(0,0);  
 **this**.**velocity** = **new** Vector2(0f,0f);  
 **this**.**boundingBox** = **new** Rectangle();  
 **this**.**json** = **new** Json();  
 **boundingBoxLocation** = BoundingBoxLocation.***BOTTOM\_LEFT***;  
 }  
  
 **public abstract void** update(Entity entity, MapManager mapMgr, **float** delta);  
  
 **protected boolean** isCollisionWithMapEntities(Entity entity, MapManager mapMgr){  
 Array<Entity> entities = mapMgr.getCurrentMapEntities();  
 **boolean** isCollisionWithMapEntities = **false**;  
  
 **for**(Entity mapEntity: entities){  
 *//Check for testing against self* **if**( mapEntity.equals(entity) ){  
 *//jump to the update statement* **continue**;  
 }  
  
 Rectangle targetRect = mapEntity.getCurrentBoundingBox();  
 **if** (**boundingBox**.overlaps(targetRect) ){  
 *//Collision* entity.sendMessage(MESSAGE.***COLLISION\_WITH\_ENTITY***);  
 isCollisionWithMapEntities = **true**;  
 **break**;  
 }  
 }  
 **return** isCollisionWithMapEntities;  
 }  
  
 **protected boolean** isCollision(Entity entitySource, Entity entityTarget){  
 **boolean** isCollisionWithMapEntities = **false**;  
  
 **if**( entitySource.equals(entityTarget) ){  
 **return false**;  
 }  
  
 **if** (entitySource.getCurrentBoundingBox().overlaps(entityTarget.getCurrentBoundingBox()) ){  
 *//Collision* entitySource.sendMessage(MESSAGE.***COLLISION\_WITH\_ENTITY***);  
 isCollisionWithMapEntities = **true**;  
 }  
  
 **return** isCollisionWithMapEntities;  
 }  
  
 **protected boolean** isCollisionWithMapLayer(Entity entity, MapManager mapMgr){  
 MapLayer mapCollisionLayer = mapMgr.getCollisionLayer();  
  
 **if**( mapCollisionLayer == **null** ){  
 **return false**;  
 }  
  
 Rectangle rectangle = **null**;  
  
 **for**( MapObject object: mapCollisionLayer.getObjects()){  
 **if**(object **instanceof** RectangleMapObject) {  
 rectangle = ((RectangleMapObject)object).getRectangle();  
 **if**( **boundingBox**.overlaps(rectangle) ){  
 *//Collision* entity.sendMessage(MESSAGE.***COLLISION\_WITH\_MAP***);  
 **return true**;  
 }  
 }  
 }  
  
 **return false**;  
 }  
  
 **protected void** setNextPositionToCurrent(Entity entity){  
 **this**.**currentEntityPosition**.**x** = **nextEntityPosition**.**x**;  
 **this**.**currentEntityPosition**.**y** = **nextEntityPosition**.**y**;  
  
 *//Gdx.app.debug(TAG, "SETTING Current Position " + tools.getEntityConfig().getEntityID()  
 // + ": (" + currentEntityPosition.x + "," + currentEntityPosition.y + ")");* entity.sendMessage(MESSAGE.***CURRENT\_POSITION***, **json**.toJson(**currentEntityPosition**));  
 }  
  
 **protected void** calculateNextPosition(**float** deltaTime){  
 **if**( **currentDirection** == **null** ) **return**;  
  
 **float** testX = **currentEntityPosition**.**x**;  
 **float** testY = **currentEntityPosition**.**y**;  
  
 **velocity**.scl(deltaTime);  
  
 **switch** (**currentDirection**) {  
 **case *LEFT*** :  
 testX -= **velocity**.**x**;  
 **break**;  
 **case *RIGHT*** :  
 testX += **velocity**.**x**;  
 **break**;  
 **case *UP*** :  
 testY += **velocity**.**y**;  
 **break**;  
 **case *DOWN*** :  
 testY -= **velocity**.**y**;  
 **break**;  
 **case *DOWN\_LEFT*** :  
 testX -= **velocity**.**x**;  
 testY -= **velocity**.**y**;  
 **break**;  
 **case *DOWN\_RIGHT*** :  
 testX += **velocity**.**x**;  
 testY -= **velocity**.**y**;  
 **break**;  
 **case *UP\_LEFT*** :  
 testX -= **velocity**.**x**;  
 testY += **velocity**.**y**;  
 **break**;  
 **case *UP\_RIGHT*** :  
 testX += **velocity**.**x**;  
 testY += **velocity**.**y**;  
 **break**;  
 **default**:  
 **break**;  
 }  
  
 **nextEntityPosition**.**x** = testX;  
 **nextEntityPosition**.**y** = testY;  
  
 *//velocity* **velocity**.scl(1 / deltaTime);  
 }  
  
 **protected void** initBoundingBox(**float** widthPercentageToReduce, **float** heightPercentageToReduce){  
 *//Update the current bounding box* **float** width;  
 **float** height;  
  
 **if**( widthPercentageToReduce > 0 && widthPercentageToReduce < 1){  
 width = Entity.FRAME\_WIDTH \* widthPercentageToReduce;  
 }**else**{  
 width = Entity.FRAME\_WIDTH;  
 }  
  
 **if**( heightPercentageToReduce > 0 && heightPercentageToReduce < 1){  
 height = Entity.FRAME\_HEIGHT \* heightPercentageToReduce;  
 }**else**{  
 height = Entity.FRAME\_HEIGHT;  
 }  
  
 **if**( width == 0 || height == 0){  
 Gdx.app.debug(TAG, **"Width and Height are 0!! "** + width + **":"** + height);  
 }  
  
 *//Need to account for the unitscale, since the map coordinates will be in pixels* **float** minX;  
 **float** minY;  
  
 **float** origWidth = Entity.FRAME\_WIDTH;  
 **float** origHeight = Entity.FRAME\_HEIGHT;  
  
 **if**( Map.UNIT\_SCALE > 0 ) {  
 minX = nextEntityPosition.x / Map.UNIT\_SCALE;  
 minY = nextEntityPosition.y / Map.UNIT\_SCALE;  
 }**else**{  
 minX = nextEntityPosition.x;  
 minY = nextEntityPosition.y;  
 }  
  
  
 boundingBox.setWidth(width);  
 boundingBox.setHeight(height);  
  
 **switch**(boundingBoxLocation){  
 **case** BOTTOM\_LEFT:  
 boundingBox.set(minX, minY, width, height);  
 **break**;  
 **case** BOTTOM\_CENTER:  
 boundingBox.setCenter(minX + origWidth/2, minY + origHeight/4);  
 **break**;  
 **case** CENTER:  
 boundingBox.setCenter(minX + origWidth/2, minY + origHeight/2);  
 **break**;  
 }  
 }  
  
 **protected void** updateBoundingBoxPosition(Vector2 position){  
 *//Need to account for the unit\_scale, since the map coordinates will be in pixels* **float** minX;  
 **float** minY;  
  
 **if**( Map.UNIT\_SCALE > 0 ) {  
 minX = position.x / Map.UNIT\_SCALE;  
 minY = position.y / Map.UNIT\_SCALE;  
 }**else**{  
 minX = position.x;  
 minY = position.y;  
 }  
  
 **switch**(boundingBoxLocation){  
 **case** BOTTOM\_LEFT:  
 boundingBox.set(minX, minY, boundingBox.getWidth(), boundingBox.getHeight());  
 **break**;  
 **case** BOTTOM\_CENTER:  
 boundingBox.setCenter(minX + Entity.FRAME\_WIDTH/2, minY + Entity.FRAME\_HEIGHT/4);  
 **break**;  
 **case** CENTER:  
 boundingBox.setCenter(minX + Entity.FRAME\_WIDTH/2, minY + Entity.FRAME\_HEIGHT/2);  
 **break**;  
 }  
 }  
}

**package** com.mygdx.game.components;  
  
**import** com.badlogic.gdx.graphics.Camera;  
**import** com.badlogic.gdx.graphics.g2d.Animation;  
**import** com.badlogic.gdx.graphics.g2d.Batch;  
**import** com.badlogic.gdx.math.GridPoint2;  
**import** com.badlogic.gdx.math.Vector2;  
**import** com.badlogic.gdx.utils.Array;  
**import** com.mygdx.game.tools.Entity;  
**import** com.mygdx.game.tools.EntityConfig;  
**import** com.mygdx.game.tools.EntityConfig.AnimationConfig;  
**import** com.mygdx.game.maps.MapManager;  
  
  
**public class** PlayerGraphicsComponent **extends** GraphicsComponent {  
  
 **private static final** String ***TAG*** = PlayerGraphicsComponent.**class**.getSimpleName();  
  
 **protected** Vector2 **previousPosition**;  
  
 **public** PlayerGraphicsComponent(){  
 **previousPosition** = **new** Vector2(0,0);  
 }  
  
 @Override  
 **public void** receiveMessage(String message) {  
 *//Gdx.app.debug(TAG, "Got message " + message);* String[] string = message.split(***MESSAGE\_TOKEN***);  
  
 **if**( string.**length** == 0 ) **return**;  
  
 *//Specifically for messages with 1 object payload* **if**( string.**length** == 2 ) {  
 **if** (string[0].equalsIgnoreCase(MESSAGE.***CURRENT\_POSITION***.toString())) {  
 **currentPosition** = **json**.fromJson(Vector2.**class**, string[1]);  
 } **else if** (string[0].equalsIgnoreCase(MESSAGE.***INIT\_START\_POSITION***.toString())) {  
 **currentPosition** = **json**.fromJson(Vector2.**class**, string[1]);  
 } **else if** (string[0].equalsIgnoreCase(MESSAGE.***CURRENT\_STATE***.toString())) {  
 **currentState** = **json**.fromJson(Entity.State.**class**, string[1]);  
 } **else if** (string[0].equalsIgnoreCase(MESSAGE.***CURRENT\_DIRECTION***.toString())) {  
 **currentDirection** = **json**.fromJson(Entity.Direction.**class**, string[1]);  
 } **else if** (string[0].equalsIgnoreCase(MESSAGE.***LOAD\_ANIMATIONS***.toString())) {  
 EntityConfig entityConfig = **json**.fromJson(EntityConfig.**class**, string[1]);  
 Array<AnimationConfig> animationConfigs = entityConfig.getAnimationConfig();  
  
 **for**( AnimationConfig animationConfig : animationConfigs ){  
 Array<String> textureNames = animationConfig.getTexturePaths();  
 Array<GridPoint2> points = animationConfig.getGridPoints();  
 Entity.AnimationType animationType = animationConfig.getAnimationType();  
 **float** frameDuration = animationConfig.getFrameDuration();  
 Animation animation = **null**;  
  
 **if**( textureNames.**size** == 1) {  
 animation = loadAnimation(textureNames.get(0), points, frameDuration);  
 }**else if**( textureNames.**size** == 2){  
 animation = loadAnimation(textureNames.get(0), textureNames.get(1), points, frameDuration);  
 }  
  
 **animations**.put(animationType, animation);  
 }  
 }  
 }  
 }  
  
 @Override  
 **public void** update(Entity entity, MapManager mapMgr, Batch batch, **float** delta){  
 updateAnimations(delta);  
  
 *//Player has moved* **if**( **previousPosition**.**x** != **currentPosition**.**x** ||  
 **previousPosition**.**y** != **currentPosition**.**y**){  
 notify(**""**, ComponentObserver.ComponentEvent.***PLAYER\_HAS\_MOVED***);  
 **previousPosition** = **currentPosition**.cpy();  
 }  
  
  
 Camera camera = mapMgr.getCamera();  
 camera.**position**.set(**currentPosition**.**x**, **currentPosition**.**y**, 0f);  
 camera.update();  
  
 batch.begin();  
 batch.draw(**currentFrame**, **currentPosition**.**x**, **currentPosition**.**y**, 1.2f, 2);  
 batch.end();  
  
 }  
  
 @Override  
 **public void** dispose(){  
 }  
  
}

**package** com.mygdx.game.components;  
  
**import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.Input;  
**import** com.badlogic.gdx.InputProcessor;  
**import** com.badlogic.gdx.math.Vector3;  
**import** com.mygdx.game.tools.Entity;  
**import** com.mygdx.game.screens.MainGameScreen;  
  
**public class** PlayerInputComponent **extends** InputComponent **implements** InputProcessor {  
  
 **private final static** String ***TAG*** = PlayerInputComponent.**class**.getSimpleName();  
 **private** Vector3 **lastMouseCoordinates**;  
  
 **public** PlayerInputComponent(){  
 **this**.**lastMouseCoordinates** = **new** Vector3();  
 }  
  
 @Override  
 **public void** dispose() {  
 Gdx.*input*.setInputProcessor(**null**);  
 }  
  
 @Override  
 **public void** receiveMessage(String message) {  
 String[] string = message.split(***MESSAGE\_TOKEN***);  
  
 **if**( string.**length** == 0 ) **return**;  
  
 *//Specifically for messages with 1 object payload* **if**( string.**length** == 2 ) {  
 **if** (string[0].equalsIgnoreCase(MESSAGE.***CURRENT\_DIRECTION***.toString())) {  
 **currentDirection** = **json**.fromJson(Entity.Direction.**class**, string[1]);  
 }  
 }  
 }  
  
 @Override  
 **public void** update(Entity entity, **float** delta){  
 *//Keyboard input* **if**(*keys*.get(Keys.***PAUSE***)) {  
 System.***out***.println(**"INPUT PAUSED"**);  
 MainGameScreen.*setGameState*(MainGameScreen.GameState.***PAUSED***);  
 pauseReleased();  
 }  
 **else if**( *keys*.get(Keys.***LEFT***)){  
 entity.sendMessage(MESSAGE.***CURRENT\_STATE***, **json**.toJson(Entity.State.***WALKING***));  
 entity.sendMessage(MESSAGE.***CURRENT\_DIRECTION***, **json**.toJson(Entity.Direction.***LEFT***));  
 }**else if**( *keys*.get(Keys.***RIGHT***)){  
 entity.sendMessage(MESSAGE.***CURRENT\_STATE***, **json**.toJson(Entity.State.***WALKING***));  
 entity.sendMessage(MESSAGE.***CURRENT\_DIRECTION***, **json**.toJson(Entity.Direction.***RIGHT***));  
 }**else if**( *keys*.get(Keys.***UP***)){  
 entity.sendMessage(MESSAGE.***CURRENT\_STATE***, **json**.toJson(Entity.State.***WALKING***));  
 entity.sendMessage(MESSAGE.***CURRENT\_DIRECTION***, **json**.toJson(Entity.Direction.***UP***));  
 }**else if**(*keys*.get(Keys.***DOWN***)){  
 entity.sendMessage(MESSAGE.***CURRENT\_STATE***, **json**.toJson(Entity.State.***WALKING***));  
 entity.sendMessage(MESSAGE.***CURRENT\_DIRECTION***, **json**.toJson(Entity.Direction.***DOWN***));  
 } **else if**( *keys*.get(Keys.***UP\_LEFT***)){  
 entity.sendMessage(MESSAGE.***CURRENT\_STATE***, **json**.toJson(Entity.State.***WALKING***));  
 entity.sendMessage(MESSAGE.***CURRENT\_DIRECTION***, **json**.toJson(Entity.Direction.***UP\_LEFT***));  
 }**else if**( *keys*.get(Keys.***UP\_RIGHT***)){  
 entity.sendMessage(MESSAGE.***CURRENT\_STATE***, **json**.toJson(Entity.State.***WALKING***));  
 entity.sendMessage(MESSAGE.***CURRENT\_DIRECTION***, **json**.toJson(Entity.Direction.***UP\_RIGHT***));  
 } **else if**(*keys*.get(Keys.***DOWN\_LEFT***)){  
 entity.sendMessage(MESSAGE.***CURRENT\_STATE***, **json**.toJson(Entity.State.***WALKING***));  
 entity.sendMessage(MESSAGE.***CURRENT\_DIRECTION***, **json**.toJson(Entity.Direction.***DOWN\_LEFT***));  
 } **else if**(*keys*.get(Keys.***DOWN\_RIGHT***)){  
 entity.sendMessage(MESSAGE.***CURRENT\_STATE***, **json**.toJson(Entity.State.***WALKING***));  
 entity.sendMessage(MESSAGE.***CURRENT\_DIRECTION***, **json**.toJson(Entity.Direction.***DOWN\_RIGHT***));  
 } **else if**(*keys*.get(Keys.***QUIT***)) {  
 quitReleased();  
 Gdx.*app*.exit();  
 } **else**{  
 entity.sendMessage(MESSAGE.***CURRENT\_STATE***, **json**.toJson(Entity.State.***IDLE***));  
 **if**( **currentDirection** == **null** ){  
 entity.sendMessage(MESSAGE.***CURRENT\_DIRECTION***, **json**.toJson(Entity.Direction.***DOWN***));  
 }  
 }  
  
 *//Mouse input* **if**( *mouseButtons*.get(Mouse.***SELECT***)) {  
 *//Gdx.app.debug(TAG, "Mouse LEFT click at : (" + lastMouseCoordinates.x + "," + lastMouseCoordinates.y + ")" );* entity.sendMessage(MESSAGE.***INIT\_SELECT\_ENTITY***, **json**.toJson(**lastMouseCoordinates**));  
 *mouseButtons*.put(Mouse.***SELECT***, **false**);  
 }  
 }  
  
 @Override  
 **public boolean** keyDown(**int** keycode) {  
 **if**( keycode == Input.Keys.***LEFT*** || keycode == Input.Keys.***A***){  
 **this**.leftPressed();  
 }  
 **if**( keycode == Input.Keys.***RIGHT*** || keycode == Input.Keys.***D***){  
 **this**.rightPressed();  
 }  
 **if**( keycode == Input.Keys.***UP*** || keycode == Input.Keys.***W***){  
 **this**.upPressed();  
 }  
 **if**( keycode == Input.Keys.***DOWN*** || keycode == Input.Keys.***S***){  
 **this**.downPressed();  
 }  
  
 **if**( keycode == Input.Keys.***Y***){  
 **this**.upLeftPressed();  
  
 }**if**( keycode == Input.Keys.***I***){  
 **this**.upRightPressed();  
  
 }**if**( keycode == Input.Keys.***N***){  
 **this**.downLeftPressed();  
  
 }**if**( keycode == Input.Keys.***COMMA***){  
 **this**.downRightPressed();  
 }  
  
 **if**( keycode == Input.Keys.***Q***){  
 **this**.quitPressed();  
 }  
 **if**( keycode == Input.Keys.***P*** ){  
 **this**.pausePressed();  
 }  
  
 **return true**;  
 }  
  
 @Override  
 **public boolean** keyUp(**int** keycode) {  
 **if**( keycode == Input.Keys.***LEFT*** || keycode == Input.Keys.***A***){  
 **this**.leftReleased();  
 }  
 **if**( keycode == Input.Keys.***RIGHT*** || keycode == Input.Keys.***D***){  
 **this**.rightReleased();  
 }  
 **if**( keycode == Input.Keys.***UP*** || keycode == Input.Keys.***W***){  
 **this**.upReleased();  
 }  
 **if**( keycode == Input.Keys.***DOWN*** || keycode == Input.Keys.***S***){  
 **this**.downReleased();  
 }  
  
  
 **if**( keycode == Input.Keys.***Y***){  
 **this**.upLeftReleased();  
  
 }**if**( keycode == Input.Keys.***I***){  
 **this**.upRightReleased();  
  
 }**if**( keycode == Input.Keys.***N***){  
 **this**.downLeftReleased();  
  
 }**if**( keycode == Input.Keys.***COMMA***){  
 **this**.downRightReleased();  
 }  
  
  
 **if**( keycode == Input.Keys.***Q***){  
 **this**.quitReleased();  
 }  
 **if**( keycode == Input.Keys.***P*** ){  
 **this**.pauseReleased();  
 }  
 **return true**;  
 }  
  
 @Override  
 **public boolean** keyTyped(**char** character) {  
 **return false**;  
 }  
  
 @Override  
 **public boolean** touchDown(**int** screenX, **int** screenY, **int** pointer, **int** button) {  
 *//Gdx.app.debug(TAG, "GameScreen: MOUSE DOWN........: (" + screenX + "," + screenY + ")" );* **if**( button == Input.Buttons.***LEFT*** || button == Input.Buttons.***RIGHT*** ){  
 **this**.setClickedMouseCoordinates(screenX, screenY);  
 }  
  
 *//left is selection, right is context menu* **if**( button == Input.Buttons.***LEFT***){  
 **this**.selectMouseButtonPressed(screenX, screenY);  
 }  
 **if**( button == Input.Buttons.***RIGHT***){  
 **this**.doActionMouseButtonPressed(screenX, screenY);  
 }  
 **return true**;  
 }  
  
 @Override  
 **public boolean** touchUp(**int** screenX, **int** screenY, **int** pointer, **int** button) {  
 *//left is selection, right is context menu* **if**( button == Input.Buttons.***LEFT***){  
 **this**.selectMouseButtonReleased(screenX, screenY);  
 }  
 **if**( button == Input.Buttons.***RIGHT***){  
 **this**.doActionMouseButtonReleased(screenX, screenY);  
 }  
 **return true**;  
 }  
  
 @Override  
 **public boolean** touchDragged(**int** screenX, **int** screenY, **int** pointer) {  
 **return false**;  
 }  
  
 @Override  
 **public boolean** mouseMoved(**int** screenX, **int** screenY) {  
 **return false**;  
 }  
  
 @Override  
 **public boolean** scrolled(**int** amount) {  
 **return false**;  
 }  
   
 *//Key presses* **public void** leftPressed(){  
 *keys*.put(Keys.***LEFT***, **true**);  
 }  
   
 **public void** rightPressed(){  
 *keys*.put(Keys.***RIGHT***, **true**);  
 }  
   
 **public void** upPressed(){  
 *keys*.put(Keys.***UP***, **true**);  
 }  
   
 **public void** downPressed(){  
 *keys*.put(Keys.***DOWN***, **true**);  
 }  
  
 **public void** upRightPressed(){  
 *keys*.put(Keys.***UP\_RIGHT***, **true**);  
 }  
  
 **public void** upLeftPressed(){  
 *keys*.put(Keys.***UP\_LEFT***, **true**);  
 }  
  
 **public void** downRightPressed(){  
 *keys*.put(Keys.***DOWN\_RIGHT***, **true**);  
 }  
  
 **public void** downLeftPressed(){  
 *keys*.put(Keys.***DOWN\_LEFT***, **true**);  
 }  
  
 **public void** quitPressed(){  
 *keys*.put(Keys.***QUIT***, **true**);  
 }  
  
 **public void** pausePressed() {  
 *keys*.put(Keys.***PAUSE***, **true**);  
 }  
   
 **public void** setClickedMouseCoordinates(**int** x,**int** y){  
 **lastMouseCoordinates**.set(x, y, 0);  
 }  
   
 **public void** selectMouseButtonPressed(**int** x, **int** y){  
 *mouseButtons*.put(Mouse.***SELECT***, **true**);  
 }  
   
 **public void** doActionMouseButtonPressed(**int** x, **int** y){  
 *mouseButtons*.put(Mouse.***DOACTION***, **true**);  
 }  
   
 *//Releases* **public void** leftReleased(){  
 *keys*.put(Keys.***LEFT***, **false**);  
 }  
   
 **public void** rightReleased(){  
 *keys*.put(Keys.***RIGHT***, **false**);  
 }  
   
 **public void** upReleased(){  
 *keys*.put(Keys.***UP***, **false**);  
 }  
   
 **public void** downReleased(){  
 *keys*.put(Keys.***DOWN***, **false**);  
 }  
  
 **public void** upRightReleased(){  
 *keys*.put(Keys.***UP\_RIGHT***, **false**);  
 }  
  
 **public void** upLeftReleased(){  
 *keys*.put(Keys.***UP\_LEFT***, **false**);  
 }  
  
 **public void** downRightReleased(){  
 *keys*.put(Keys.***DOWN\_RIGHT***, **false**);  
 }  
  
 **public void** downLeftReleased(){  
 *keys*.put(Keys.***DOWN\_LEFT***, **false**);  
 }  
  
 **public void** quitReleased(){  
 *keys*.put(Keys.***QUIT***, **false**);  
 }  
  
 **public void** pauseReleased() { *keys*.put(Keys.***PAUSE***, **false**);}  
  
 **public void** selectMouseButtonReleased(**int** x, **int** y){  
 *mouseButtons*.put(Mouse.***SELECT***, **false**);  
 }  
   
 **public void** doActionMouseButtonReleased(**int** x, **int** y){  
 *mouseButtons*.put(Mouse.***DOACTION***, **false**);  
 }  
  
 **public static void** hide(){  
 *keys*.put(Keys.***LEFT***, **false**);  
 *keys*.put(Keys.***RIGHT***, **false**);  
 *keys*.put(Keys.***UP***, **false**);  
 *keys*.put(Keys.***DOWN***, **false**);  
 *keys*.put(Keys.***QUIT***, **false**);  
 }  
  
}

**package** com.mygdx.game.components;  
  
**import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.graphics.Camera;  
**import** com.badlogic.gdx.maps.MapLayer;  
**import** com.badlogic.gdx.maps.MapObject;  
**import** com.badlogic.gdx.maps.objects.RectangleMapObject;  
**import** com.badlogic.gdx.math.Rectangle;  
**import** com.badlogic.gdx.math.Vector2;  
**import** com.badlogic.gdx.math.Vector3;  
**import** com.badlogic.gdx.math.collision.Ray;  
**import** com.badlogic.gdx.utils.Array;  
**import** com.mygdx.game.tools.Entity;  
**import** com.mygdx.game.maps.Map;  
**import** com.mygdx.game.maps.MapFactory;  
**import** com.mygdx.game.maps.MapManager;  
  
**public class** PlayerPhysicsComponent **extends** PhysicsComponent {  
  
 **private static final** String ***TAG*** = PlayerPhysicsComponent.**class**.getSimpleName();  
  
 **private** Entity.State **state**;  
 **private** Vector3 **mouseSelectCoordinates**;  
 **private boolean isMouseSelectEnabled** = **false**;  
 **private** Ray **selectionRay**;  
 **private float selectRayMaximumDistance** = 30.0f;  
 **private** String **previousEnemySpawn**;  
  
 **public** PlayerPhysicsComponent(){  
 **boundingBoxLocation** = BoundingBoxLocation.***BOTTOM\_CENTER***;  
 initBoundingBox(0.6f, 0.4f);  
 **super**.**velocity**.set(8f,8f);  
 **previousEnemySpawn** = **"0"**;  
  
 **mouseSelectCoordinates** = **new** Vector3(0,0,0);  
 **selectionRay** = **new** Ray(**new** Vector3(), **new** Vector3());  
 }  
  
 @Override  
 **public void** dispose(){  
 }  
  
 @Override  
 **public void** receiveMessage(String message) {  
 *//Gdx.app.debug(TAG, "Got message " + message);* String[] string = message.split(Component.***MESSAGE\_TOKEN***);  
  
 **if**( string.**length** == 0 ) **return**;  
  
 *//Specifically for messages with 1 object payload* **if**( string.**length** == 2 ) {  
 **if** (string[0].equalsIgnoreCase(MESSAGE.***INIT\_START\_POSITION***.toString())) {  
 **currentEntityPosition** = **json**.fromJson(Vector2.**class**, string[1]);  
 **nextEntityPosition**.set(**currentEntityPosition**.**x**, **currentEntityPosition**.**y**);  
 **previousEnemySpawn** = **"0"**;  
 notify(**previousEnemySpawn**, ComponentObserver.ComponentEvent.***ENEMY\_SPAWN\_LOCATION\_CHANGED***);  
 } **else if** (string[0].equalsIgnoreCase(MESSAGE.***CURRENT\_STATE***.toString())) {  
 **state** = **json**.fromJson(Entity.State.**class**, string[1]);  
 } **else if** (string[0].equalsIgnoreCase(MESSAGE.CURRENT\_DIRECTION.toString())) {  
 currentDirection = json.fromJson(Entity.Direction.**class**, string[1]);  
 } **else if** (string[0].equalsIgnoreCase(MESSAGE.INIT\_SELECT\_ENTITY.toString())) {  
 mouseSelectCoordinates = json.fromJson(Vector3.**class**, string[1]);  
 isMouseSelectEnabled = **true**;  
 }  
 }  
 }  
  
 @Override  
 **public void** update(Entity entity, MapManager mapMgr, **float** delta) {  
 *//We want the hitbox to be at the feet for a better feel* updateBoundingBoxPosition(nextEntityPosition);  
 updatePortalLayerActivation(mapMgr);  
 updateEnemySpawnLayerActivation(mapMgr);  
  
 **if**(isMouseSelectEnabled){  
 selectMapEntityCandidate(mapMgr);  
 isMouseSelectEnabled = **false**;  
 }  
  
 **if** (!isCollisionWithMapLayer(entity, mapMgr) &&  
 !isCollisionWithMapEntities(entity, mapMgr) &&  
 state == Entity.State.WALKING){  
 setNextPositionToCurrent(entity);  
  
 Camera camera = mapMgr.getCamera();  
 camera.position.set(currentEntityPosition.x, currentEntityPosition.y, 0f);  
 camera.update();  
 }**else**{  
 updateBoundingBoxPosition(currentEntityPosition);  
 }  
  
 calculateNextPosition(delta);  
 }  
  
 **private void** selectMapEntityCandidate(MapManager mapMgr){  
 Array<Entity> currentEntities = mapMgr.getCurrentMapEntities();  
  
 *//Convert screen coordinates to world coordinates, then to unit scale coordinates* mapMgr.getCamera().unproject(mouseSelectCoordinates);  
 mouseSelectCoordinates.x /= Map.UNIT\_SCALE;  
 mouseSelectCoordinates.y /= Map.UNIT\_SCALE;  
  
 *//Gdx.app.debug(TAG, "Mouse Coordinates " + "(" + mouseSelectCoordinates.x + "," + mouseSelectCoordinates.y + ")");* **for**( Entity mapEntity : currentEntities ) {  
 *//Don't break, reset all entities* mapEntity.sendMessage(MESSAGE.ENTITY\_DESELECTED);  
 Rectangle mapEntityBoundingBox = mapEntity.getCurrentBoundingBox();  
 *//Gdx.app.debug(TAG, "Entity Candidate Location " + "(" + mapEntityBoundingBox.x + "," + mapEntityBoundingBox.y + ")");* **if** (mapEntity.getCurrentBoundingBox().contains(mouseSelectCoordinates.x, mouseSelectCoordinates.y)) {  
 *//Check distance* selectionRay.set(boundingBox.x, boundingBox.y, 0.0f, mapEntityBoundingBox.x, mapEntityBoundingBox.y, 0.0f);  
 **float** distance = selectionRay.origin.dst(selectionRay.direction);  
  
 **if**( distance <= selectRayMaximumDistance){  
 *//We have a valid tools selection  
 //Picked/Selected* Gdx.app.debug(TAG, **"Selected Entity! "** + mapEntity.getEntityConfig().getEntityID());  
 mapEntity.sendMessage(MESSAGE.ENTITY\_SELECTED);  
 }  
 }  
 }  
 }  
  
 **private boolean** updateEnemySpawnLayerActivation(MapManager mapMgr){  
 MapLayer mapEnemySpawnLayer = mapMgr.getEnemySpawnLayer();  
  
 **if**( mapEnemySpawnLayer == **null** ){  
 **return false**;  
 }  
  
 Rectangle rectangle = **null**;  
  
 **for**( MapObject object: mapEnemySpawnLayer.getObjects()){  
 **if**(object **instanceof** RectangleMapObject) {  
 rectangle = ((RectangleMapObject)object).getRectangle();  
  
 **if** (boundingBox.overlaps(rectangle) ){  
 String enemySpawnID = object.getName();  
  
 **if**( enemySpawnID == **null** ) {  
 **return false**;  
 }  
  
 **if**( previousEnemySpawn.equalsIgnoreCase(enemySpawnID) ){  
 *//Gdx.app.debug(TAG, "Enemy Spawn Area already activated " + enemySpawnID);* **return true**;  
 }**else**{  
 Gdx.app.debug(TAG, **"Enemy Spawn Area "** + enemySpawnID + **" Activated with previous Spawn value: "** + previousEnemySpawn);  
 previousEnemySpawn = enemySpawnID;  
 }  
  
 notify(enemySpawnID, ComponentObserver.ComponentEvent.ENEMY\_SPAWN\_LOCATION\_CHANGED);  
 **return true**;  
 }  
 }  
 }  
  
 *//If no collision, reset the value* **if**( !previousEnemySpawn.equalsIgnoreCase(String.valueOf(0)) ){  
 Gdx.app.debug(TAG, **"Enemy Spawn Area RESET with previous value "** + previousEnemySpawn);  
 previousEnemySpawn = String.valueOf(0);  
 notify(previousEnemySpawn, ComponentObserver.ComponentEvent.ENEMY\_SPAWN\_LOCATION\_CHANGED);  
 }  
  
 **return false**;  
 }  
  
 **private boolean** updatePortalLayerActivation(MapManager mapMgr){  
 MapLayer mapPortalLayer = mapMgr.getPortalLayer();  
  
 **if**( mapPortalLayer == **null** ){  
 Gdx.app.debug(TAG, **"Portal Layer doesn't exist!"**);  
 **return false**;  
 }  
  
 Rectangle rectangle = **null**;  
  
 **for**( MapObject object: mapPortalLayer.getObjects()){  
  
 **if**(object **instanceof** RectangleMapObject) {  
 rectangle = ((RectangleMapObject)object).getRectangle();  
  
 **if** (boundingBox.overlaps(rectangle) ){  
  
 *//TODO speak about this,* String mapName;  
 String specificPortal;  
 String temp = object.getName();  
 **if**( temp == **null**) {  
 **return false**;  
 }  
  
 **if**(temp.contains(**"."**)) {  
 mapName = temp.substring(0, temp.indexOf(**'.'**));  
 specificPortal = temp.substring(temp.indexOf(**'.'**)+1);  
 Gdx.app.log(TAG, **"mapName is "** + mapName);  
 Gdx.app.log(TAG, **"specificPortal is "** + specificPortal);  
 }  
 **else** {  
 specificPortal = **"PLAYER\_STAR"**;  
 mapName = temp;  
 }  
  
 mapMgr.setSpecificPortal(specificPortal);  
 mapMgr.loadMap(MapFactory.MapType.valueOf(mapName));  
  
  
 currentEntityPosition.x = mapMgr.getPlayerStartUnitScaled().x;  
 currentEntityPosition.y = mapMgr.getPlayerStartUnitScaled().y;  
 nextEntityPosition.x = mapMgr.getPlayerStartUnitScaled().x;  
 nextEntityPosition.y = mapMgr.getPlayerStartUnitScaled().y;  
  
 *//mapMgr.setClosestStartPositionFromScaledUnits(currentEntityPosition);* Gdx.app.debug(TAG, **"Portal Activated"**);  
 **return true**;  
 }  
 }  
 }  
 **return false**;  
 }  
  
}

## src.com.mygdx.game.gui;

**package** com.mygdx.game.gui;  
  
**import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.graphics.Texture;  
**import** com.badlogic.gdx.graphics.g2d.TextureRegion;  
**import** com.badlogic.gdx.math.MathUtils;  
**import** com.badlogic.gdx.scenes.scene2d.InputEvent;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Image;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Table;  
**import** com.badlogic.gdx.scenes.scene2d.ui.TextButton;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Window;  
**import** com.badlogic.gdx.scenes.scene2d.utils.ClickListener;  
**import** com.badlogic.gdx.scenes.scene2d.utils.TextureRegionDrawable;  
**import** com.mygdx.game.battle.BattleObserver;  
**import** com.mygdx.game.battle.BattleState;  
**import** com.mygdx.game.japanese.KanaLetter;  
**import** com.mygdx.game.japanese.KanaLettersFactory;  
**import** com.mygdx.game.japanese.KanjiLetter;  
**import** com.mygdx.game.japanese.KanjiLettersFactory;  
**import** com.mygdx.game.tools.Utility;  
  
**import** java.util.ArrayList;  
  
**public class** BattleUI **extends** Window **implements** BattleObserver {  
  
 *//****TODO speak about this,* private final static** String ***TAG*** = BattleUI.**class**.getSimpleName();  
  
 **private** KanjiLetter **kanjiToAnswer**;  
 **private** KanaLetter **kanaToAnswer**;  
 **private** ArrayList<KanaLetter> **kanaLettersList**;  
 **private** ArrayList<KanjiLetter> **kanjiLettersList**;  
   
 **private** BattleState **battleState** = **null**;  
 **private final float checkTimer** = 2.5f;  
 **private float battleTimer** = 0;  
  
 **private** Image **hiro**;  
 **private** Texture **texture**;  
  
 **private** Table **topTable**;  
 **private** Table bottomTable;  
 **private** Image **imageToAnswer**;  
  
 **private** Table **innerTableOne**;  
 **private** Table **innerTableTwo**;  
 **private** Table **innerTableThree**;  
 **private** Table **innerTableFour**;  
 **private float padBetweenTables**;  
  
 **private** String **firstInnerTableEquivalent**;  
 **private** String **secondInnerTableEquivalent**;  
 **private** String **thirdInnerTableEquivalent**;  
 **private** String **fourthInnerTableEquivalent**;  
   
 **private** KanaLetter **equivalentImageOptionOne**;  
 **private** KanaLetter **equivalentImageOptionTwo**;  
 **private** KanaLetter **equivalentImageOptionThree**;  
 **private** KanaLetter **equivalentImageOptionFour**;  
 **private float padBetweenImages**;  
  
 **private** TextButton **runButton** = **null**;  
  
 **public** BattleUI(){  
 **super**(**""**, Utility.*GUI\_SKINS*, **"solid\_background"**);  
   
 **battleTimer** = 0;  
 **battleState** = **new** BattleState();  
 **battleState**.addObserver(**this**);  
  
 **hiro** = **new** Image(Utility.*HIRO\_TEXTUREATLAS*.findRegion(**"hiro\_left"**));  
 **kanaLettersList** = KanaLettersFactory.*getInstance*().getKanaLettersList();  
 **kanjiLettersList** = KanjiLettersFactory.*getInstance*().getKanjiLettersList();  
  
 **imageToAnswer** = **new** Image();  
  
 **padBetweenTables** = **this**.getWidth()/5;  
 **innerTableOne** = **new** Table();  
 **innerTableTwo** = **new** Table();  
 **innerTableThree** = **new** Table();  
 **innerTableFour** = **new** Table();  
  
  
 **padBetweenImages** = **this**.getWidth()/3;  
 **equivalentImageOptionOne** = **new** KanaLetter();  
 **equivalentImageOptionTwo** = **new** KanaLetter();  
 **equivalentImageOptionThree** = **new** KanaLetter();  
 **equivalentImageOptionFour** = **new** KanaLetter();  
  
 **topTable** = **new** Table();  
 **topTable**.add(**imageToAnswer**).padRight(**this**.getWidth()/2);  
 **topTable**.add(**hiro**).padLeft(**this**.getWidth()/2);  
  
 bottomTable = **new** Table();  
 **runButton** = **new** TextButton(**"Run"**, Utility.*GUI\_SKINS*);  
  
 **this**.add(**topTable**).expand().fill().padTop(**this**.getHeight()/1.5f);;  
 **this**.row();  
 **this**.add(bottomTable).padBottom(**this**.getHeight()/4);;  
 **this**.row();  
 **this**.add(**runButton**).left().padBottom(**this**.getHeight()/3).padLeft(**this**.getWidth()/8);  
  
 **runButton**.addListener(  
 **new** ClickListener() {  
 @Override  
 **public void** clicked(InputEvent event, **float** x, **float** y) {  
 **battleState**.playerRuns();  
 }  
 }  
 );  
 }  
  
 **public void** battleZoneTriggered(**int** battleZoneValue){  
 **battleState**.setCurrentZoneLevel(battleZoneValue);  
  
 **switch** (battleZoneValue){  
 **case** 1:  
 **texture** = **new** Texture(Gdx.*files*.internal(**"sprites/maps/forest.png"**));  
 **this**.background(**new** TextureRegionDrawable(**new** TextureRegion(**texture**)));  
 **break**;  
 **case** 2:  
 **texture** = **new** Texture(Gdx.*files*.internal(**"sprites/maps/cave.png"**));  
 **this**.background(**new** TextureRegionDrawable(**new** TextureRegion(**texture**)));  
 **break**;  
 **case** 3:  
 **texture** = **new** Texture(Gdx.*files*.internal(**"sprites/maps/coast.png"**));  
 **this**.background(**new** TextureRegionDrawable(**new** TextureRegion(**texture**)));  
 **break**;  
 **case** 4:  
 **texture** = **new** Texture(Gdx.*files*.internal(**"sprites/maps/ice\_forest.png"**));  
 **this**.background(**new** TextureRegionDrawable(**new** TextureRegion(**texture**)));  
 **break**;  
 **case** 5:  
 **texture** = **new** Texture(Gdx.*files*.internal(**"sprites/maps/desert\_temple.png"**));  
 **this**.background(**new** TextureRegionDrawable(**new** TextureRegion(**texture**)));  
 **break**;  
 **case** 6:  
 **texture** = **new** Texture(Gdx.*files*.internal(**"sprites/maps/lava.png"**));  
 **this**.background(**new** TextureRegionDrawable(**new** TextureRegion(**texture**)));  
 **break**;  
 **default**: **break**;  
 }  
  
 }  
  
 **public boolean** isBattleReady(){  
 **if**( **battleTimer** > **checkTimer**){  
 Gdx.*app*.log(***TAG***, **"battle is ready"**);  
 **battleTimer** = 0;  
 **return battleState**.isOpponentReady();  
 }**else**{  
 **return false**;  
 }  
 }  
  
 **public** BattleState getCurrentState(){  
 **return battleState**;  
 }  
  
 @Override  
 **public void** onNotify(String letterToAnswer, BattleEvent event) {  
 **int** randomVal;  
 **int** randomChoice;  
 KanjiLetter tempKanji;  
 **switch**(event){  
 **case *KANJI\_ADDED***:  
 *//question* Gdx.*app*.log(***TAG***, **"Kanji ToAnswer is "** + letterToAnswer);  
 **kanjiToAnswer** = KanjiLettersFactory.*getInstance*().getKanjiLetter(letterToAnswer);  
 **this**.**imageToAnswer**.setDrawable(**new** TextureRegionDrawable(Utility.*LARGE\_KANJI\_TEXTUREATLAS*.findRegion(**kanjiToAnswer**.getKanjiNameID())));  
  
 bottomTable.clear();  
 **innerTableOne**.clear();  
 **innerTableTwo**.clear();  
 **innerTableThree**.clear();  
 **innerTableFour**.clear();  
  
 *//answer options* Image equivalent;  
 String splitPortion;  
 **int** indexOfFullStop;  
 randomChoice = MathUtils.*random*(1,2);  
  
  
  
 **firstInnerTableEquivalent** = **kanjiToAnswer**.getHiraganaEquivalent();  
 **for** (**int** j = 0; j < **firstInnerTableEquivalent**.length(); j++) {  
 **if** (**firstInnerTableEquivalent**.charAt(j) == **'.'**) {  
 **try** {  
 indexOfFullStop = **firstInnerTableEquivalent**.indexOf(**'.'**, j + 1);  
 splitPortion = **firstInnerTableEquivalent**.substring(j + 1, indexOfFullStop);  
 **if**(randomChoice == 1) {  
 equivalent = **new** Image(Utility.*SMALL\_HIRAGANA\_TEXTUREATLAS*.findRegion(splitPortion));  
 } **else** {  
 **kanaToAnswer** = KanaLettersFactory.*getInstance*().getHiraganaLetter(splitPortion);  
 equivalent = **new** Image(Utility.*SMALL\_KATAKANA\_TEXTUREATLAS*.  
 findRegion(**kanaToAnswer**.getKatakanaEquivalent()));  
 }  
 **innerTableOne**.add(equivalent).left();  
 } **catch** (Exception e) {  
 }  
  
 }  
 }  
  
 randomVal = MathUtils.*random*(0,29);  
 tempKanji = **kanjiLettersList**.get(randomVal);  
  
 **secondInnerTableEquivalent** = tempKanji.getHiraganaEquivalent();  
 **for** (**int** j = 0; j < **secondInnerTableEquivalent**.length(); j++) {  
 **if** (**secondInnerTableEquivalent**.charAt(j) == **'.'**) {  
 **try** {  
 indexOfFullStop = **secondInnerTableEquivalent**.indexOf(**'.'**, j + 1);  
 splitPortion = **secondInnerTableEquivalent**.substring(j + 1, indexOfFullStop);  
 **if**(randomChoice == 1) {  
 equivalent = **new** Image(Utility.*SMALL\_HIRAGANA\_TEXTUREATLAS*.findRegion(splitPortion));  
 } **else** {  
 **kanaToAnswer** = KanaLettersFactory.*getInstance*().getHiraganaLetter(splitPortion);  
 equivalent = **new** Image(Utility.*SMALL\_KATAKANA\_TEXTUREATLAS*.  
 findRegion(**kanaToAnswer**.getKatakanaEquivalent()));  
 }  
 **innerTableTwo**.add(equivalent).left();  
 } **catch** (Exception e) {  
 }  
  
 }  
 }  
  
 randomVal = MathUtils.*random*(0,29);  
 tempKanji = **kanjiLettersList**.get(randomVal);  
  
 **thirdInnerTableEquivalent** = tempKanji.getHiraganaEquivalent();  
 **for** (**int** j = 0; j < **thirdInnerTableEquivalent**.length(); j++) {  
 **if** (**thirdInnerTableEquivalent**.charAt(j) == **'.'**) {  
 **try** {  
 indexOfFullStop = **thirdInnerTableEquivalent**.indexOf(**'.'**, j + 1);  
 splitPortion = **thirdInnerTableEquivalent**.substring(j + 1, indexOfFullStop);  
 **if**(randomChoice == 1) {  
 equivalent = **new** Image(Utility.*SMALL\_HIRAGANA\_TEXTUREATLAS*.findRegion(splitPortion));  
 } **else** {  
 **kanaToAnswer** = KanaLettersFactory.*getInstance*().getHiraganaLetter(splitPortion);  
 equivalent = **new** Image(Utility.*SMALL\_KATAKANA\_TEXTUREATLAS*.  
 findRegion(**kanaToAnswer**.getKatakanaEquivalent()));  
 }  
 **innerTableThree**.add(equivalent).left();  
 } **catch** (Exception e) {  
 }  
  
 }  
 }  
  
 randomVal = MathUtils.*random*(0,29);  
 tempKanji = **kanjiLettersList**.get(randomVal);  
  
 **fourthInnerTableEquivalent** = tempKanji.getHiraganaEquivalent();  
 **for** (**int** j = 0; j < **fourthInnerTableEquivalent**.length(); j++) {  
 **if** (**fourthInnerTableEquivalent**.charAt(j) == **'.'**) {  
 **try** {  
 indexOfFullStop = **fourthInnerTableEquivalent**.indexOf(**'.'**, j + 1);  
 splitPortion = **fourthInnerTableEquivalent**.substring(j + 1, indexOfFullStop);  
 **if**(randomChoice == 1) {  
 equivalent = **new** Image(Utility.*SMALL\_HIRAGANA\_TEXTUREATLAS*.findRegion(splitPortion));  
 } **else** {  
 **kanaToAnswer** = KanaLettersFactory.*getInstance*().getHiraganaLetter(splitPortion);  
 equivalent = **new** Image(Utility.*SMALL\_KATAKANA\_TEXTUREATLAS*.  
 findRegion(**kanaToAnswer**.getKatakanaEquivalent()));  
 }  
 **innerTableFour**.add(equivalent).left();  
 } **catch** (Exception e) {  
 }  
  
 }  
 }  
  
 randomizeOrder(**true**);  
 addTableListeners();  
  
 **break**;  
 **case *HIRAGANA\_ADDED***:  
 bottomTable.clear();  
 **equivalentImageOptionOne**.clear();  
 **equivalentImageOptionTwo**.clear();  
 **equivalentImageOptionThree**.clear();  
 **equivalentImageOptionFour**.clear();  
  
 Gdx.*app*.log(***TAG***, **"Kana ToAnswer is "** + letterToAnswer);  
 **kanaToAnswer** = KanaLettersFactory.*getInstance*().getHiraganaLetter(letterToAnswer);  
 **imageToAnswer**.setDrawable(**new** TextureRegionDrawable(Utility.*LARGE\_HIRAGANA\_TEXTUREATLAS*.findRegion(**kanaToAnswer**.getHiraganaEquivalent())));  
  
 **equivalentImageOptionOne** = **kanaToAnswer**;  
 **equivalentImageOptionOne**.setDrawable(**new** TextureRegionDrawable(Utility.*MEDIUM\_ROMAJI\_TEXTUREATLAS*.findRegion(**equivalentImageOptionOne**.getRomajiEquivalent())));  
 Gdx.*app*.log(***TAG***, **"equivalentImageOptionOne is "** + **equivalentImageOptionOne**.getRomajiEquivalent());  
  
 randomVal = MathUtils.*random*(0,106);  
 **equivalentImageOptionTwo** = **kanaLettersList**.get(randomVal);  
 **equivalentImageOptionTwo**.setDrawable(**new** TextureRegionDrawable(Utility.*MEDIUM\_ROMAJI\_TEXTUREATLAS*.findRegion(**equivalentImageOptionTwo**.getRomajiEquivalent())));  
 Gdx.*app*.log(***TAG***, **"equivalentImageOptionTwo is "** + **equivalentImageOptionTwo**.getRomajiEquivalent());  
  
 randomVal = MathUtils.*random*(0,106);  
 **equivalentImageOptionThree** = **kanaLettersList**.get(randomVal);  
 **equivalentImageOptionThree**.setDrawable(**new** TextureRegionDrawable(Utility.*MEDIUM\_ROMAJI\_TEXTUREATLAS*.findRegion(**equivalentImageOptionThree**.getRomajiEquivalent())));  
 Gdx.*app*.log(***TAG***, **"equivalentImageOptionThree is "** + **equivalentImageOptionThree**.getRomajiEquivalent());  
  
 randomVal = MathUtils.*random*(0,106);  
 **equivalentImageOptionFour** = **kanaLettersList**.get(randomVal);  
 **equivalentImageOptionFour**.setDrawable(**new** TextureRegionDrawable(Utility.*MEDIUM\_ROMAJI\_TEXTUREATLAS*.findRegion(**equivalentImageOptionFour**.getRomajiEquivalent())));  
 Gdx.*app*.log(***TAG***, **"equivalentImageOptionFour is "** + **equivalentImageOptionFour**.getRomajiEquivalent());  
  
 randomizeOrder(**false**);  
 addKanaImageListeners();  
  
 **break**;  
 **case *KATAKANA\_ADDED***:  
 bottomTable.clear();  
 **equivalentImageOptionOne**.clear();  
 **equivalentImageOptionTwo**.clear();  
 **equivalentImageOptionThree**.clear();  
 **equivalentImageOptionFour**.clear();  
  
 Gdx.*app*.log(***TAG***, **"Kana ToAnswer is "** + letterToAnswer);  
 **kanaToAnswer** = KanaLettersFactory.*getInstance*().getKatakanaLetter(letterToAnswer);  
 **imageToAnswer**.setDrawable(**new** TextureRegionDrawable(Utility.*LARGE\_KATAKANA\_TEXTUREATLAS*.findRegion(**kanaToAnswer**.getKatakanaEquivalent())));  
  
 **equivalentImageOptionOne** = **kanaToAnswer**;  
 **equivalentImageOptionOne**.setDrawable(**new** TextureRegionDrawable(Utility.*MEDIUM\_HIRAGANA\_TEXTUREATLAS*.findRegion(**equivalentImageOptionOne**.getHiraganaEquivalent())));  
 Gdx.*app*.log(***TAG***, **"equivalentImageOptionOne is "** + **equivalentImageOptionOne**.getHiraganaEquivalent());  
  
 randomVal = MathUtils.*random*(0,106);  
 **equivalentImageOptionTwo** = **kanaLettersList**.get(randomVal);  
 **equivalentImageOptionTwo**.setDrawable(**new** TextureRegionDrawable(Utility.*MEDIUM\_HIRAGANA\_TEXTUREATLAS*.findRegion(**equivalentImageOptionTwo**.getHiraganaEquivalent())));  
 Gdx.*app*.log(***TAG***, **"equivalentImageOptionTwo is "** + **equivalentImageOptionTwo**.getHiraganaEquivalent());  
  
 randomVal = MathUtils.*random*(0,106);  
 **equivalentImageOptionThree** = **kanaLettersList**.get(randomVal);  
 **equivalentImageOptionThree**.setDrawable(**new** TextureRegionDrawable(Utility.*MEDIUM\_HIRAGANA\_TEXTUREATLAS*.findRegion(**equivalentImageOptionThree**.getHiraganaEquivalent())));  
 Gdx.*app*.log(***TAG***, **"equivalentImageOptionThree is "** + **equivalentImageOptionThree**.getHiraganaEquivalent());  
  
 randomVal = MathUtils.*random*(0,106);  
 **equivalentImageOptionFour** = **kanaLettersList**.get(randomVal);  
 **equivalentImageOptionFour**.setDrawable(**new** TextureRegionDrawable(Utility.*MEDIUM\_HIRAGANA\_TEXTUREATLAS*.findRegion(**equivalentImageOptionFour**.getHiraganaEquivalent())));  
 Gdx.*app*.log(***TAG***, **"equivalentImageOptionFour is "** + **equivalentImageOptionFour**.getHiraganaEquivalent());  
  
 randomizeOrder(**false**);  
 addKanaImageListeners();  
  
 **break**;  
 **default**:  
 **break**;  
 }  
 }  
  
 @Override  
 **public void** act(**float** delta){  
 **battleTimer** = (**battleTimer** + delta)%60;  
 *//Gdx.app.log(TAG, "battleTimer is " + battleTimer);* **super**.act(delta);  
 }  
  
 **public void** randomizeOrder(**boolean** tables){  
 **int** randomVal = MathUtils.*random*(0,3);  
 **switch** (randomVal){  
 **case** 0:  
 **if**(tables) {  
 bottomTable.add(**innerTableOne**);  
 bottomTable.add(**innerTableTwo**).padLeft(**padBetweenTables**);  
 bottomTable.add(**innerTableThree**).padLeft(**padBetweenTables**);  
 bottomTable.add(**innerTableFour**).padLeft(**padBetweenTables**);  
 } **else** {  
 bottomTable.add(**equivalentImageOptionOne**);  
 bottomTable.add(**equivalentImageOptionTwo**).padLeft(**padBetweenImages**);  
 bottomTable.add(**equivalentImageOptionThree**).padLeft(**padBetweenImages**);  
 bottomTable.add(**equivalentImageOptionFour**).padLeft(**padBetweenImages**);  
 }  
 **break**;  
 **case** 1:  
 **if**(tables) {  
 bottomTable.add(**innerTableTwo**);  
 bottomTable.add(**innerTableOne**).padLeft(**padBetweenTables**);  
 bottomTable.add(**innerTableThree**).padLeft(**padBetweenTables**);  
 bottomTable.add(**innerTableFour**).padLeft(**padBetweenTables**).padRight(**padBetweenTables**);  
 } **else** {  
 bottomTable.add(**equivalentImageOptionTwo**);  
 bottomTable.add(**equivalentImageOptionOne**).padLeft(**padBetweenImages**);  
 bottomTable.add(**equivalentImageOptionThree**).padLeft(**padBetweenImages**);  
 bottomTable.add(**equivalentImageOptionFour**).padLeft(**padBetweenImages**);  
 }  
 **break**;  
 **case** 2:  
 **if**(tables) {  
 bottomTable.add(**innerTableTwo**);  
 bottomTable.add(**innerTableThree**).padLeft(**padBetweenTables**);  
 bottomTable.add(**innerTableOne**).padLeft(**padBetweenTables**);  
 bottomTable.add(**innerTableFour**).padLeft(**padBetweenTables**);  
 } **else** {  
 bottomTable.add(**equivalentImageOptionTwo**);  
 bottomTable.add(**equivalentImageOptionThree**).padLeft(**padBetweenImages**);  
 bottomTable.add(**equivalentImageOptionOne**).padLeft(**padBetweenImages**);  
 bottomTable.add(**equivalentImageOptionFour**).padLeft(**padBetweenImages**);  
 }  
 **break**;  
 **case** 3:  
 **if**(tables) {  
 bottomTable.add(**innerTableTwo**);  
 bottomTable.add(**innerTableThree**).padLeft(**padBetweenTables**);  
 bottomTable.add(**innerTableFour**).padLeft(**padBetweenTables**);  
 bottomTable.add(**innerTableOne**).padLeft(**padBetweenTables**);  
 }**else** {  
 bottomTable.add(**equivalentImageOptionTwo**);  
 bottomTable.add(**equivalentImageOptionThree**).padLeft(**padBetweenImages**);  
 bottomTable.add(**equivalentImageOptionFour**).padLeft(**padBetweenImages**);  
 bottomTable.add(**equivalentImageOptionOne**).padLeft(**padBetweenImages**);  
 }  
 **break**;  
 **default**: **break**;  
 }  
  
 }  
  
 **public void** addTableListeners(){  
 **innerTableOne**.addListener(  
 **new** ClickListener() {  
 @Override  
 **public void** clicked(InputEvent event, **float** x, **float** y) {  
 Gdx.*app*.log(***TAG***, **firstInnerTableEquivalent**);  
 **if**(**firstInnerTableEquivalent**.equalsIgnoreCase(**kanjiToAnswer**.getHiraganaEquivalent()))  
 **battleState**.answeredCorrectly(**kanjiToAnswer**.getKanjiNameID());  
 **else** {  
 **battleState**.answeredIncorrectly(**kanjiToAnswer**.getKanjiNameID());  
 }  
  
 }  
 }  
 );  
 **innerTableTwo**.addListener(  
 **new** ClickListener() {  
 @Override  
 **public void** clicked(InputEvent event, **float** x, **float** y) {  
 Gdx.*app*.log(***TAG***, **secondInnerTableEquivalent**);  
 **if**(**secondInnerTableEquivalent**.equalsIgnoreCase(**kanjiToAnswer**.getHiraganaEquivalent()))  
 **battleState**.answeredCorrectly(**kanjiToAnswer**.getKanjiNameID());  
 **else** {  
 **battleState**.answeredIncorrectly(**kanjiToAnswer**.getKanjiNameID());  
 }  
 }  
 }  
 );  
 **innerTableThree**.addListener(  
 **new** ClickListener() {  
 @Override  
 **public void** clicked(InputEvent event, **float** x, **float** y) {  
 Gdx.*app*.log(***TAG***, **thirdInnerTableEquivalent**);  
 **if**(**thirdInnerTableEquivalent**.equalsIgnoreCase(**kanjiToAnswer**.getHiraganaEquivalent()))  
 **battleState**.answeredCorrectly(**kanjiToAnswer**.getKanjiNameID());  
 **else** {  
 **battleState**.answeredIncorrectly(**kanjiToAnswer**.getKanjiNameID());  
 }  
 }  
 }  
 );  
 **innerTableFour**.addListener(  
 **new** ClickListener() {  
 @Override  
 **public void** clicked(InputEvent event, **float** x, **float** y) {  
 Gdx.*app*.log(***TAG***, **fourthInnerTableEquivalent**);  
 **if**(**fourthInnerTableEquivalent**.equalsIgnoreCase(**kanjiToAnswer**.getHiraganaEquivalent()))  
 **battleState**.answeredCorrectly(**kanjiToAnswer**.getKanjiNameID());  
 **else** {  
 **battleState**.answeredIncorrectly(**kanjiToAnswer**.getKanjiNameID());  
 }  
 }  
 }  
 );  
  
 }  
  
 **public void** addKanaImageListeners(){  
 **equivalentImageOptionOne**.addListener(  
 **new** ClickListener() {  
 @Override  
 **public void** clicked(InputEvent event, **float** x, **float** y) {  
 **if**(**equivalentImageOptionOne**.getHiraganaEquivalent().equalsIgnoreCase(**kanaToAnswer**.getHiraganaEquivalent()))  
 **battleState**.answeredCorrectly(**kanaToAnswer**.getHiraganaEquivalent());  
 **else** {  
 **battleState**.answeredIncorrectly(**kanaToAnswer**.getHiraganaEquivalent());  
 }  
  
 }  
 }  
 );  
 **equivalentImageOptionTwo**.addListener(  
 **new** ClickListener() {  
 @Override  
 **public void** clicked(InputEvent event, **float** x, **float** y) {;  
 **if**(**equivalentImageOptionTwo**.getHiraganaEquivalent().equalsIgnoreCase(**kanaToAnswer**.getHiraganaEquivalent()))  
 **battleState**.answeredCorrectly(**kanaToAnswer**.getHiraganaEquivalent());  
 **else** {  
 **battleState**.answeredIncorrectly(**kanaToAnswer**.getHiraganaEquivalent());  
 }  
 }  
 }  
 );  
 **equivalentImageOptionThree**.addListener(  
 **new** ClickListener() {  
 @Override  
 **public void** clicked(InputEvent event, **float** x, **float** y) {  
 **if**(**equivalentImageOptionThree**.getHiraganaEquivalent().equalsIgnoreCase(**kanaToAnswer**.getHiraganaEquivalent()))  
 **battleState**.answeredCorrectly(**kanaToAnswer**.getHiraganaEquivalent());  
 **else** {  
 **battleState**.answeredIncorrectly(**kanaToAnswer**.getHiraganaEquivalent());  
 }  
 }  
 }  
 );  
 **equivalentImageOptionFour**.addListener(  
 **new** ClickListener() {  
 @Override  
 **public void** clicked(InputEvent event, **float** x, **float** y) {  
 **if**(**equivalentImageOptionFour**.getHiraganaEquivalent().equalsIgnoreCase(**kanaToAnswer**.getHiraganaEquivalent()))  
 **battleState**.answeredCorrectly(**kanaToAnswer**.getHiraganaEquivalent());  
 **else** {  
 **battleState**.answeredIncorrectly(**kanaToAnswer**.getHiraganaEquivalent());  
 }  
 }  
 }  
 );  
  
 }  
  
}

**package** com.mygdx.game.gui;  
  
**public interface** InventoryObserver {  
 **public static enum** InventoryEvent {  
 ***ITEM\_CONSUMED***,  
 ***ADD\_WAND\_AP***,  
 ***NONE*** }  
  
 **void** onNotify(**final** String value, InventoryEvent event);  
}

**package** com.mygdx.game.gui;  
  
**public interface** InventorySubject {  
 **public void** addObserver(InventoryObserver inventoryObserver);  
 **public void** removeObserver(InventoryObserver inventoryObserver);  
 **public void** removeAllObservers();  
 **public void** notify(**final** String value, InventoryObserver.InventoryEvent event);  
}

**package** com.mygdx.game.gui;  
  
**import** com.badlogic.gdx.Screen;  
**import** com.badlogic.gdx.scenes.scene2d.InputEvent;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Cell;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Label;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Table;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Window;  
**import** com.badlogic.gdx.scenes.scene2d.utils.ClickListener;  
**import** com.badlogic.gdx.utils.Align;  
**import** com.badlogic.gdx.utils.Array;  
**import** com.mygdx.game.components.Component;  
**import** com.mygdx.game.tools.Utility;  
**import** com.mygdx.game.inventory.InventoryItem;  
**import** com.mygdx.game.inventory.InventoryItemFactory;  
**import** com.mygdx.game.inventory.InventoryItemLocation;  
**import** com.mygdx.game.inventory.InventoryItem.ItemNameID;  
**import** com.mygdx.game.inventory.InventorySlot;  
  
**public class** InventoryUI **extends** Window **implements** InventorySubject {  
  
 **private final static** String ***TAG*** = InventoryUI.**class**.getSimpleName();  
  
 **private int numSlots** = 35;  
 **private int lengthSlotRow** = 7;  
 **private** Table **inventorySlotTable**;  
  
 **private** InventorySlot **topSlot**;  
 **private** InventorySlot **slotToRemove**;  
 **private** Label **itemDescription**;  
 **private** String **description**;  
  
 **private float slotWidth**;  
 **private float slotHeight**;  
  
 **float menuItemWindowWidth**;  
 **float menuItemWindowHeight**;  
  
 **private** Array<InventoryObserver> **\_observers**;  
  
 **public** InventoryUI(**float** width, **float** height) {  
 **super**(**"Inventory"**, Utility.*GUI\_SKINS*);  
 **this**.getTitleLabel().setAlignment(Align.***center***);  
  
 **\_observers** = **new** Array<InventoryObserver>();  
  
 **menuItemWindowWidth** = width;  
 **menuItemWindowHeight** = height;  
  
 **this**.pad(**this**.getPadTop() + **menuItemWindowHeight** / 15, 10,  
 **menuItemWindowHeight** / 30, 10);  
  
  
 **slotWidth** = **menuItemWindowWidth** / 8;  
 **slotHeight** = **menuItemWindowHeight** / 7.5f;  
  
 **description** = **""**;  
 **itemDescription** = **new** Label(**""**, Utility.*GUI\_SKINS*, **"list\_text"**);  
 **topSlot** = **new** InventorySlot();  
 **inventorySlotTable** = **new** Table();  
 **inventorySlotTable**.setName(**"Inventory\_Slot\_Table"**);  
 *//inventorySlotTable.setPosition(menuItemWindowWidth/1.5f,menuItemWindowHeight/2.6f);* **for** (**int** i = 1; i <= **numSlots**; i++) {  
 InventorySlot inventorySlot = **new** InventorySlot();  
  
 **inventorySlotTable**.add(inventorySlot).size(**slotWidth**, **slotHeight**);  
  
 inventorySlot.addListener(**new** ClickListener() {  
 @Override  
 **public void** touchUp(InputEvent event, **float** x, **float** y, **int** pointer, **int** button) {  
 **super**.touchUp(event, x, y, pointer, button);  
  
 **slotToRemove** = (InventorySlot) event.getListenerActor();  
 **if** (**slotToRemove**.hasItem()) {  
 **topSlot**.clearAllInventoryItems();  
 InventoryItem item = **slotToRemove**.getTopInventoryItem();  
 InventoryItem item2 = InventoryItemFactory.*getInstance*()  
 .getInventoryItem(item.getItemNameID());  
  
 **topSlot**.addActor(item);  
 **slotToRemove**.addActor(item2);  
  
 *//Gdx.app.debug(TAG, "item.getItemNameID is: " + item.getItemNameID().toString());* **description** = item.getItemShortDescription();  
  
 **int** mid;  
 **int** spaceAfterMid;  
 mid = **description**.length()/2;  
 spaceAfterMid = **description**.indexOf(**" "**, mid);  
 **itemDescription**.setText(**description**.substring(0, spaceAfterMid) + **"\n"** +  
 **description**.substring(spaceAfterMid));  
  
 }  
 }  
  
 }  
 );  
  
 **if** (i % **lengthSlotRow** == 0) {  
 **inventorySlotTable**.row();  
 }  
 }  
  
 **topSlot**.addListener(**new** ClickListener() {  
 @Override  
 **public void** touchUp(InputEvent event, **float** x, **float** y, **int** pointer, **int** button) {  
 **super**.touchUp(event, x, y, pointer, button);  
  
 InventorySlot slot = (InventorySlot) event.getListenerActor();  
 **if** (slot.hasItem()) {  
 InventoryItem item = slot.getTopInventoryItem();  
 **if** (item.isConsumable()) {  
 String itemInfo = item.getItemUseType() + Component.***MESSAGE\_TOKEN*** + item.getItemUseValue();  
 InventoryUI.**this**.notify(itemInfo, InventoryObserver.InventoryEvent.***ITEM\_CONSUMED***);  
 **topSlot**.clearAllInventoryItems();  
 **slotToRemove**.clearAllInventoryItems();  
 **itemDescription**.setText(**""**);  
 }  
 }  
 }  
  
  
 }  
 );  
  
 **topSlot**.setPosition(**menuItemWindowWidth** / 16f, **menuItemWindowHeight** / 1.35f);  
 **topSlot**.setSize(**menuItemWindowWidth** / 6, **menuItemWindowHeight** /7f);  
 **itemDescription**.setPosition(**menuItemWindowWidth** / 4f, **menuItemWindowHeight** / 1.25f);  
 **inventorySlotTable**.setPosition(**menuItemWindowWidth** / 2f, **menuItemWindowHeight** / 2.6f);  
  
 **this**.addActor(**topSlot**);  
 **this**.addActor(**itemDescription**);  
 **this**.addActor(**inventorySlotTable**);  
  
 **this**.setSize(**menuItemWindowWidth**, **menuItemWindowHeight**);  
 *//this.debug();* }  
  
 **public void** updateSize(**float** width, **float** height) {  
 **float** newMenuItemWindowWidth = width;  
 **float** newMenuItemWindowHeight = height;  
  
 **topSlot**.setPosition(newMenuItemWindowWidth / 16f, newMenuItemWindowHeight / 1.35f);  
 **topSlot**.setSize(newMenuItemWindowWidth / 6, newMenuItemWindowHeight /7f);  
 **itemDescription**.setPosition(newMenuItemWindowWidth / 4f, newMenuItemWindowHeight / 1.25f);  
 **inventorySlotTable**.setPosition(newMenuItemWindowWidth / 2f, newMenuItemWindowHeight / 2.6f);  
  
 **for** (**int** i = 0; i <= **inventorySlotTable**.getCells().**size** - 1; i++) {  
 **inventorySlotTable**.getCells().get(i).size(newMenuItemWindowWidth / 8, newMenuItemWindowHeight / 7.5f);  
 }  
  
 **this**.setSize(newMenuItemWindowWidth, newMenuItemWindowHeight);  
 }  
  
  
 **public** Table getInventorySlotTable() {  
 **return inventorySlotTable**;  
 }  
  
 **public static** Array<InventoryItemLocation> getInventory(Table targetTable) {  
 Array<Cell> cells = targetTable.getCells();  
 Array<InventoryItemLocation> items = **new** Array<InventoryItemLocation>();  
 **for** (**int** i = 0; i < cells.**size**; i++) {  
 InventorySlot inventorySlot = ((InventorySlot) cells.get(i).getActor());  
 **if** (inventorySlot == **null**) **continue**;  
 **int** numItems = inventorySlot.getNumItems();  
 **if** (numItems > 0) {  
 items.add(**new** InventoryItemLocation(  
 i, inventorySlot.getTopInventoryItem().getItemNameID().toString()));  
 }  
 }  
 **return** items;  
 }  
  
 **public static void** populateInventory(Table targetTable, Array<InventoryItemLocation> inventoryItems) {  
 clearInventoryItems(targetTable);  
  
 Array<Cell> cells = targetTable.getCells();  
 **for**(**int** i = 0; i < inventoryItems.size; i++){  
 InventoryItemLocation itemLocation = inventoryItems.get(i);  
 ItemNameID itemNameID = ItemNameID.valueOf(itemLocation.getItemNameAtLocation());  
 InventorySlot inventorySlot = ((InventorySlot)cells.get(itemLocation.getLocationIndex()).getActor());  
 InventoryItem item = InventoryItemFactory.getInstance().getInventoryItem(itemNameID);  
 inventorySlot.add(item);  
 }  
 }  
  
 **public static void** clearInventoryItems(Table targetTable){  
 Array<Cell> cells = targetTable.getCells();  
 **for**( **int** i = 0; i < cells.**size**; i++){  
 InventorySlot inventorySlot = (InventorySlot)cells.get(i).getActor();  
 **if**( inventorySlot == **null** ) **continue**;  
 inventorySlot.clearAllInventoryItems();  
 }  
 }  
  
  
 @Override  
 **public void** addObserver(InventoryObserver inventoryObserver) {  
 **\_observers**.add(inventoryObserver);  
 }  
  
 @Override  
 **public void** removeObserver(InventoryObserver inventoryObserver) {  
 **\_observers**.removeValue(inventoryObserver, **true**);  
 }  
  
 @Override  
 **public void** removeAllObservers() {  
 **for** (InventoryObserver observer : **\_observers**) {  
 **\_observers**.removeValue(observer, **true**);  
 }  
 }  
  
 @Override  
 **public void** notify(String value, InventoryObserver.InventoryEvent event) {  
 **for** (InventoryObserver observer : **\_observers**) {  
 observer.onNotify(value, event);  
 }  
 }  
  
  
}

**package** com.mygdx.game.gui;  
  
**import** com.badlogic.gdx.scenes.scene2d.ui.Image;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Label;  
**import** com.badlogic.gdx.scenes.scene2d.ui.ScrollPane;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Table;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Window;  
**import** com.badlogic.gdx.utils.Align;  
**import** com.mygdx.game.japanese.KanaLetter;  
**import** com.mygdx.game.japanese.KanaLettersFactory;  
**import** com.mygdx.game.tools.Utility;  
  
**import** java.util.ArrayList;  
  
**public class** KanaUI **extends** Window{  
  
 **private final static** String TAG = ProgressUI.**class**.getSimpleName();  
  
 Table table;  
 String kanaType;  
 **float** menuItemWindowWidth;  
 **float** menuItemWindowHeight;  
  
 **public** KanaUI(**float** width, **float** height, String temp){  
 **super**(temp, Utility.GUI\_SKINS);  
 **this**.getTitleLabel().setAlignment(Align.center);  
  
 menuItemWindowWidth = width;  
 menuItemWindowHeight = height;  
 kanaType = temp;  
  
 **this**.pad(**this**.getPadTop() + menuItemWindowHeight / 15, 10,  
 menuItemWindowHeight / 30, 10);  
  
 Label text;  
 Image equivalent;  
 KanaLetter kanaLetter;  
 table = **new** Table();  
  
 ArrayList<KanaLetter> kanaLettersList = KanaLettersFactory.getInstance().getKanaLettersList();  
 *//Gdx.app.log(TAG, "kanaLettersList size is " + kanaLettersList.size() );* **for**(**int** i = 0; i < kanaLettersList.size(); i++){  
  
 **if**(**this**.kanaType.equalsIgnoreCase(**"hiragana"**)) {  
 kanaLetter = kanaLettersList.get(i);  
 equivalent = **new** Image(Utility.MEDIUM\_HIRAGANA\_TEXTUREATLAS.findRegion(kanaLetter.getHiraganaEquivalent()));  
 table.add(equivalent).left();  
 text = **new** Label(**"romaji"** + **"\n"** + **"Equivalent: "**, Utility.GUI\_SKINS, **"list\_text"**);  
 table.add(text).left();  
 equivalent = **new** Image(Utility.SMALL\_ROMAJI\_TEXTUREATLAS.findRegion(kanaLetter.getRomajiEquivalent()));  
 table.add(equivalent).left();  
 text = **new** Label(**"katakana"** + **"\n"** + **"Equivalent: "**, Utility.GUI\_SKINS, **"list\_text"**);  
 table.add(text);  
 equivalent = **new** Image(Utility.SMALL\_KATAKANA\_TEXTUREATLAS.findRegion(kanaLetter.getKatakanaEquivalent()));  
 table.add(equivalent).left();  
 }  
 **else** {  
 kanaLetter = kanaLettersList.get(i);  
 equivalent = **new** Image(Utility.MEDIUM\_KATAKANA\_TEXTUREATLAS.findRegion(kanaLetter.getKatakanaEquivalent()));  
 table.add(equivalent).left();  
 text = **new** Label(**"romaji"** + **"\n"** + **"Equivalent: "**, Utility.GUI\_SKINS, **"list\_text"**);  
 table.add(text).left();  
 equivalent = **new** Image(Utility.SMALL\_ROMAJI\_TEXTUREATLAS.findRegion(kanaLetter.getRomajiEquivalent()));  
 table.add(equivalent).left();  
 text = **new** Label(**"hiragana"** + **"\n"** + **"Equivalent: "**, Utility.GUI\_SKINS, **"list\_text"**);  
 table.add(text);  
 equivalent = **new** Image(Utility.SMALL\_HIRAGANA\_TEXTUREATLAS.findRegion(kanaLetter.getHiraganaEquivalent()));  
 table.add(equivalent).left();  
 }  
  
 table.row();  
 }  
  
 ScrollPane scrollPane = **new** ScrollPane(table);  
 **this**.add(scrollPane).fill().expand();  
  
 **this**.setSize(menuItemWindowWidth, menuItemWindowHeight);  
 *//this.debug();* }  
  
 **public void** updateSize(**float** width, **float** height){  
 **float** newMenuItemWindowWidth = width;  
 **float** newMenuItemWindowHeight = height;  
  
 **this**.setSize(newMenuItemWindowWidth, newMenuItemWindowHeight);  
 }  
  
  
}

**package** com.mygdx.game.gui;  
  
**import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Image;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Label;  
**import** com.badlogic.gdx.scenes.scene2d.ui.ScrollPane;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Table;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Window;  
**import** com.badlogic.gdx.utils.Align;  
**import** com.mygdx.game.japanese.KanjiLetter;  
**import** com.mygdx.game.japanese.KanjiLettersFactory;  
**import** com.mygdx.game.tools.Utility;  
  
**import** java.util.ArrayList;  
  
**public class** KanjiUI **extends** Window{  
  
 **private final static** String ***TAG*** = KanjiUI.**class**.getSimpleName();  
  
 Table **table**;  
 Table **innerTable**;  
 **float menuItemWindowWidth**;  
 **float menuItemWindowHeight**;  
  
 **public** KanjiUI(**float** width, **float** height) {  
 **super**(**"Kanji"**, Utility.*GUI\_SKINS*);  
 **this**.getTitleLabel().setAlignment(Align.***center***);  
  
 **menuItemWindowWidth** = width;  
 **menuItemWindowHeight** = height;  
  
 **this**.pad(**this**.getPadTop() + **menuItemWindowHeight** / 15, 10,  
 **menuItemWindowHeight** / 30, 10);  
  
 Label text;  
 String temp;  
 String split;  
 **int** indexOfFullStop;  
 Image equivalent;  
 **innerTable** = **new** Table();  
 **table** = **new** Table();  
  
 ArrayList<KanjiLetter> kanjiLettersList = KanjiLettersFactory.*getInstance*().getKanjiLettersList();  
  
 *//****TODO speak about this,* for** (**int** i = 0; i < kanjiLettersList.size(); i++) {  
  
 **if** (i == 0) {  
 text = **new** Label(**"Forest Map"**, Utility.*GUI\_SKINS*);  
 **table**.add(text).left();  
 **table**.row();  
 } **else if** (i == 5) {  
 **table**.add(**innerTable**).left();  
 **table**.row();  
  
 text = **new** Label(**"Cave Map"**, Utility.*GUI\_SKINS*);  
 **table**.add(text).left();  
 **table**.row();  
  
 **innerTable** = **new** Table();  
 } **else if** (i == 10) {  
 **table**.add(**innerTable**).left();  
 **table**.row();  
  
 text = **new** Label(**"Coast Map"**, Utility.*GUI\_SKINS*);  
 **table**.add(text).left();  
 **table**.row();  
  
 **innerTable** = **new** Table();  
 } **else if** (i == 15) {  
 **table**.add(**innerTable**).left();  
 **table**.row();  
  
 text = **new** Label(**"Ice-Forest Map"**, Utility.*GUI\_SKINS*);  
 **table**.add(text).left();  
 **table**.row();  
  
 **innerTable** = **new** Table();  
 } **else if** (i == 20) {  
 **table**.add(**innerTable**).left();  
 **table**.row();  
  
 text = **new** Label(**"Desert Castle Map"**, Utility.*GUI\_SKINS*);  
 **table**.add(text).left();  
 **table**.row();  
  
 **innerTable** = **new** Table();  
 } **else if** (i == 25) {  
 **table**.add(**innerTable**).left();  
 **table**.row();  
  
 text = **new** Label(**"Lava Map"**, Utility.*GUI\_SKINS*);  
 **table**.add(text).left();  
 **table**.row();  
  
 **innerTable** = **new** Table();  
 } **else if** (i == 30) {  
 **table**.add(**innerTable**).left();  
 **table**.row();  
  
 **innerTable** = **new** Table();  
 }  
  
 KanjiLetter kanjiLetter = kanjiLettersList.get(i);  
  
 *//Gdx.app.log(TAG, "kanjiLetter.getKanjiNameID() is " + kanjiLetter.getKanjiNameID());* equivalent = **new** Image(Utility.*MEDIUM\_KANJI\_TEXTUREATLAS*.findRegion(kanjiLetter.getKanjiNameID()));  
 **innerTable**.add(equivalent).left();  
 text = **new** Label(**"Kanji meaning: "** + kanjiLetter.getKanjiMeaning() + **"\n"** + **"hiragana Equivalent: "**, Utility.*GUI\_SKINS*, **"list\_text"**);  
 **innerTable**.add(text).left();  
  
 *//****TODO speak about this,*** temp = kanjiLetter.getHiraganaEquivalent();  
  
 **for** (**int** j = 0; j < temp.length(); j++) {  
 **if** (temp.charAt(j) == **'.'**) {  
 **try** {  
 *//Gdx.app.log(TAG, "j is: " + j);* indexOfFullStop = temp.indexOf(**'.'**, j + 1);  
 split = temp.substring(j + 1, indexOfFullStop);  
 equivalent = **new** Image(Utility.*SMALL\_HIRAGANA\_TEXTUREATLAS*.findRegion(split));  
 **innerTable**.add(equivalent).left();  
 } **catch** (Exception e) {  
 }  
  
 }  
 }  
  
 **innerTable**.row();  
 }  
  
 ScrollPane scrollPane = **new** ScrollPane(**table**);  
 **this**.add(scrollPane).fill().expand();  
 **this**.setSize(**menuItemWindowWidth**, **menuItemWindowHeight**);  
 }  
  
 **public void** updateSize(**float** width, **float** height){  
 **float** newMenuItemWindowWidth = width;  
 **float** newMenuItemWindowHeight = height;  
  
 **this**.setSize(newMenuItemWindowWidth, newMenuItemWindowHeight);  
 }  
  
  
}

**package** com.mygdx.game.gui;  
  
**import** com.badlogic.gdx.scenes.scene2d.ui.TextButton;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Window;  
**import** com.badlogic.gdx.utils.Align;  
**import** com.mygdx.game.tools.Utility;  
  
**public class** MenuListUI **extends** Window {  
  
 **private** TextButton progressButton;  
 **private** TextButton inventoryButton;  
 **private** TextButton hiraganaButton;  
 **private** TextButton katakanaButton;  
 **private** TextButton kanjiButton;  
 **private** TextButton mnemonicsButton;  
  
 **public** MenuListUI(){  
 **super**(**"Menu"**, Utility.GUI\_SKINS);  
 **this**.getTitleLabel().setAlignment(Align.center);  
  
 progressButton = **new** TextButton(**"progress"**, Utility.GUI\_SKINS);  
 inventoryButton = **new** TextButton(**"inventory"**, Utility.GUI\_SKINS);  
 hiraganaButton = **new** TextButton(**"hiragana"**, Utility.GUI\_SKINS);  
 katakanaButton = **new** TextButton(**"katakana"**, Utility.GUI\_SKINS);  
 kanjiButton = **new** TextButton(**"kanji"**, Utility.GUI\_SKINS);  
 mnemonicsButton = **new** TextButton(**"mnemonics"**, Utility.GUI\_SKINS);  
  
 *//Add to layout* defaults().expand().fill();  
  
 *//account for the title padding* **this**.pad(**this**.getPadTop() + 40, 10, 10, 10);  
  
 **this**.add(progressButton);  
 **this**.row();  
  
 **this**.add(inventoryButton);  
 **this**.row();  
  
 **this**.add(hiraganaButton);  
 **this**.row();  
  
 **this**.add(katakanaButton);  
 **this**.row();  
  
 **this**.add(kanjiButton);  
 **this**.row();  
  
 **this**.add(mnemonicsButton);  
  
 *//this.debug();  
 //this.pack();* }  
  
 **public void** updateSize(**float** width, **float** height){  
 **float** newMenuItemWindowWidth = width;  
 **float** newMenuItemWindowHeight = height;  
  
 *//this.pad(this.getPadTop() + newMenuItemWindowHeight/20, newMenuItemWindowWidth/20,  
 // newMenuItemWindowHeight/20, newMenuItemWindowWidth/20);* **this**.setSize(newMenuItemWindowWidth, newMenuItemWindowHeight);  
 }  
  
 **public** TextButton getProgressButton() {  
 **return** progressButton;  
 }  
  
 **public** TextButton getInventoryButton() {  
 **return** inventoryButton;  
 }  
  
 **public** TextButton getHiraganaButton() {  
 **return** hiraganaButton;  
 }  
  
 **public** TextButton getKatakanaButton() {  
 **return** katakanaButton;  
 }  
  
 **public** TextButton getKanjiButton() {  
 **return** kanjiButton;  
 }  
  
 **public** TextButton getMnemonicsButton() {  
 **return** mnemonicsButton;  
 }  
}

**package** com.mygdx.game.gui;  
  
**import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Image;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Label;  
**import** com.badlogic.gdx.scenes.scene2d.ui.ScrollPane;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Table;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Window;  
**import** com.badlogic.gdx.utils.Align;  
**import** com.mygdx.game.gui.ProgressUI;  
**import** com.mygdx.game.japanese.KanjiLetter;  
**import** com.mygdx.game.japanese.KanjiLettersFactory;  
**import** com.mygdx.game.tools.Utility;  
  
**import** java.util.ArrayList;  
  
**public class** MnemonicsUI **extends** Window{  
  
 **private final static** String TAG = ProgressUI.**class**.getSimpleName();  
  
 **private** Table table;  
 **private** Table innerTable;  
 **private float** menuItemWindowWidth;  
 **private float** menuItemWindowHeight;  
  
 **public** MnemonicsUI(**float** width, **float** height){  
 **super**(**"Mnemonics"**, Utility.GUI\_SKINS);  
 **this**.getTitleLabel().setAlignment(Align.center);  
  
 menuItemWindowWidth = width;  
 menuItemWindowHeight = height;  
  
 **this**.pad(**this**.getPadTop() + menuItemWindowHeight / 15, 10,  
 menuItemWindowHeight / 30, 10);  
  
 Label text;  
 String temp;  
 String toSplit;  
 **int** indexOfFullStop;  
 Image equivalent;  
 innerTable = **new** Table();  
 table = **new** Table();  
  
 ArrayList<KanjiLetter> kanjiLettersList = KanjiLettersFactory.getInstance().getKanjiLettersList();  
  
 *//TODO speak about this,* **for**(**int** i = 0; i < kanjiLettersList.size(); i++){  
  
 **if** (i == 0) {  
 text = **new** Label(**"() - Meaning, \"\" - Reading"**, Utility.GUI\_SKINS, **"list\_text"**);  
 table.add(text).center().padRight(200);  
 table.row();  
 text = **new** Label(**"Forest Map"**, Utility.GUI\_SKINS);  
 table.add(text).left();  
 table.row();  
 } **else if** (i == 5) {  
 table.add(innerTable).left();  
 table.row();  
  
 text = **new** Label(**"Cave Map"**, Utility.GUI\_SKINS);  
 table.add(text).left();  
 table.row();  
  
 innerTable = **new** Table();  
 } **else if** (i == 10) {  
 table.add(innerTable).left();  
 table.row();  
  
 text = **new** Label(**"Coast Map"**, Utility.GUI\_SKINS);  
 table.add(text).left();  
 table.row();  
  
 innerTable = **new** Table();  
 } **else if** (i == 15) {  
 table.add(innerTable).left();  
 table.row();  
  
 text = **new** Label(**"Ice-Forest Map"**, Utility.GUI\_SKINS);  
 table.add(text).left();  
 table.row();  
  
 innerTable = **new** Table();  
 } **else if** (i == 20) {  
 table.add(innerTable).left();  
 table.row();  
  
 text = **new** Label(**"Desert Castle Map"**, Utility.GUI\_SKINS);  
 table.add(text).left();  
 table.row();  
  
 innerTable = **new** Table();  
 } **else if** (i == 25) {  
 table.add(innerTable).left();  
 table.row();  
  
 text = **new** Label(**"Lava Map"**, Utility.GUI\_SKINS);  
 table.add(text).left();  
 table.row();  
  
 innerTable = **new** Table();  
 } **else if** (i == 30) {  
 table.add(innerTable).left();  
 table.row();  
  
 innerTable = **new** Table();  
 }  
  
 KanjiLetter kanjiLetter = kanjiLettersList.get(i);  
 equivalent = **new** Image(Utility.MEDIUM\_KANJI\_TEXTUREATLAS.findRegion(kanjiLetter.getKanjiNameID()));  
 innerTable.add(equivalent).left();  
  
 toSplit = kanjiLetter.getKanjiMnemonic();  
  
 **int** mid;  
 **int** spaceAfterMid;  
 mid = toSplit.length()/2;  
 spaceAfterMid = toSplit.indexOf(**" "**, mid);  
 text = **new** Label(toSplit.substring(0, spaceAfterMid) + **"\n"** +  
 toSplit.substring(spaceAfterMid), Utility.GUI\_SKINS, **"list\_text"**);  
  
 innerTable.add(text).left();  
 innerTable.row();  
 }  
  
 ScrollPane scrollPane = **new** ScrollPane(table);  
 **this**.add(scrollPane).fill().expand();  
 **this**.setSize(menuItemWindowWidth, menuItemWindowHeight);  
 }  
  
 **public void** updateSize(**float** width, **float** height){  
 **float** newMenuItemWindowWidth = width;  
 **float** newMenuItemWindowHeight = height;  
  
 **this**.setSize(newMenuItemWindowWidth, newMenuItemWindowHeight);  
 }  
  
  
}

**package** com.mygdx.game.gui;  
  
**import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.InputMultiplexer;  
**import** com.badlogic.gdx.Screen;  
**import** com.badlogic.gdx.graphics.Camera;  
**import** com.badlogic.gdx.math.MathUtils;  
**import** com.badlogic.gdx.scenes.scene2d.InputEvent;  
**import** com.badlogic.gdx.scenes.scene2d.Stage;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Image;  
**import** com.badlogic.gdx.scenes.scene2d.ui.TextButton;  
**import** com.badlogic.gdx.scenes.scene2d.utils.ClickListener;  
**import** com.badlogic.gdx.utils.Array;  
**import** com.badlogic.gdx.utils.Json;  
**import** com.badlogic.gdx.utils.viewport.ScreenViewport;  
**import** com.badlogic.gdx.utils.viewport.Viewport;  
**import** com.mygdx.game.audio.AudioManager;  
**import** com.mygdx.game.audio.AudioObserver;  
**import** com.mygdx.game.audio.AudioSubject;  
**import** com.mygdx.game.battle.BattleObserver;  
**import** com.mygdx.game.components.Component;  
**import** com.mygdx.game.components.ComponentObserver;  
**import** com.mygdx.game.japanese.KanaLetter;  
**import** com.mygdx.game.japanese.KanaLettersFactory;  
**import** com.mygdx.game.japanese.KanjiLetter;  
**import** com.mygdx.game.japanese.KanjiLettersFactory;  
**import** com.mygdx.game.japanese.LetterLvlCounter;  
**import** com.mygdx.game.maps.MapManager;  
**import** com.mygdx.game.tools.Entity;  
**import** com.mygdx.game.tools.Utility;  
**import** com.mygdx.game.inventory.InventoryItem;  
**import** com.mygdx.game.inventory.InventoryItemLocation;  
**import** com.mygdx.game.profile.ProfileManager;  
**import** com.mygdx.game.profile.ProfileObserver;  
**import** com.mygdx.game.screens.MainGameScreen;  
**import** com.mygdx.game.inventory.InventoryItem.ItemNameID;  
  
**import** java.util.ArrayList;  
  
**public class** PlayerHUD **implements** Screen, AudioSubject, ProfileObserver, ComponentObserver, InventoryObserver, BattleObserver{  
  
 **private final static** String ***TAG*** = PlayerHUD.**class**.getSimpleName();  
  
 **private** Stage **stage**;  
 **private** Viewport **viewport**;  
 **private** Camera **camera**;  
 **private** Entity **player**;  
 **private** Json **json**;  
  
 **private** MenuListUI **menuListUI**;  
 **private** ProgressUI **progressUI**;  
 **private** InventoryUI **inventoryUI**;  
 **private** KanaUI **hiraganaUI**;  
 **private** KanaUI **katakanaUI**;  
 **private** KanjiUI **kanjiUI**;  
 **private** MnemonicsUI **mnemonicsUI**;  
 **private** BattleUI **battleUI**;  
  
 **private** TextButton **menuButton**;  
 **private** TextButton **progressButton**;  
 **private** TextButton **inventoryButton**;  
 **private** TextButton **hiraganaButton**;  
 **private** TextButton **katakanaButton**;  
 **private** TextButton **kanjiButton**;  
 **private** TextButton **mnemonicsButton**;  
  
 **private** Array<Image> **all\_health\_heart**;  
 **private** Image **health\_heart**;  
  
 **private float menuItemsXaxis**;  
 **private float menuItemsYaxis**;  
 **private float menuItemWindowWidth**;  
 **private float menuItemWindowHeight**;  
  
 **private int maxNumberOfHearts** = -1;  
 **private int numberOfHearts** = -1;  
 **private** MapManager **mapManager**;  
  
 **private** Array<AudioObserver> **observers**;  
  
 **public** PlayerHUD(Camera camera, **final** Entity player, **final** InputMultiplexer multiplexer, MapManager mapMgr) {  
 **this**.**camera** = camera;  
 **this**.**player** = player;  
 **viewport** = **new** ScreenViewport(**this**.**camera**);  
 **stage** = **new** Stage(**viewport**);  
 **json** = **new** Json();  
 **mapManager** = mapMgr;  
  
 **observers** = **new** Array<AudioObserver>();  
  
 multiplexer.addProcessor(**this**.getStage());  
 multiplexer.addProcessor(player.getInputProcessor());  
 Gdx.*input*.setInputProcessor(multiplexer);  
  
 **menuItemsXaxis** = 0;  
 **menuItemsYaxis** = **stage**.getHeight()/40;  
 **menuItemWindowWidth** = **stage**.getWidth()/1.4f;  
 **menuItemWindowHeight** = **stage**.getHeight()/1.05f;  
  
 **all\_health\_heart** = **new** Array<Image>();  
 *//****TODO speak about this* for** (**int** i = 0; i<10; i++) {  
 **health\_heart** = **new** Image(Utility.*ITEMS\_TEXTUREATLAS*.findRegion(**"HEALTH\_HEART"**));  
 **health\_heart**.setPosition(**health\_heart**.getWidth() \* **all\_health\_heart**.**size** + i, **stage**.getHeight() - **health\_heart**.getHeight());  
 **health\_heart**.setVisible(**false**);  
 **all\_health\_heart**.add(**health\_heart**);  
 }  
  
  
 **menuButton** = **new** TextButton(**"menu"**, Utility.*GUI\_SKINS*);  
 **menuButton**.setPosition(**stage**.getWidth()/1.2f, **stage**.getHeight()/12);  
 **menuButton**.setVisible(**true**);  
  
 **menuListUI** = **new** MenuListUI();  
 **menuListUI**.setSize(**stage**.getWidth()/3.4f, **stage**.getHeight()/1.4f);  
 **menuListUI**.setPosition(**stage**.getWidth()/1.27f, **stage**.getHeight()/2);  
 **menuListUI**.setMovable(**false**);  
 **menuListUI**.setVisible(**false**);  
  
 **progressUI** = **new** ProgressUI(**menuItemWindowWidth**, **menuItemWindowHeight**);  
 **progressUI**.setPosition(**menuItemsXaxis**, **menuItemsYaxis**);  
 **progressUI**.setVisible(**false**);  
 **progressUI**.setMovable(**false**);  
 **progressUI**.updateTable();  
  
 *//Gdx.app.debug(TAG, "All hiragana memorised is " + LetterLvlCounter.areAllHiraganaMemorised());* **inventoryUI** = **new** InventoryUI(**menuItemWindowWidth**, **menuItemWindowHeight**);  
 **inventoryUI**.setPosition(**menuItemsXaxis**, **menuItemsYaxis**);  
 **inventoryUI**.setMovable(**false**);  
 **inventoryUI**.setVisible(**false**);  
  
 **hiraganaUI** = **new** KanaUI(**menuItemWindowWidth**, **menuItemWindowHeight**, **"Hiragana"**);  
 **hiraganaUI**.setPosition(**menuItemsXaxis**, **menuItemsYaxis**);  
 **hiraganaUI**.setMovable(**false**);  
 **hiraganaUI**.setVisible(**false**);  
  
 **katakanaUI** = **new** KanaUI(**menuItemWindowWidth**, **menuItemWindowHeight**, **"Katakana"**);  
 **katakanaUI**.setPosition(**menuItemsXaxis**, **menuItemsYaxis**);  
 **katakanaUI**.setMovable(**false**);  
 **katakanaUI**.setVisible(**false**);  
  
 **kanjiUI** = **new** KanjiUI(**menuItemWindowWidth**, **menuItemWindowHeight**);  
 **kanjiUI**.setPosition(**menuItemsXaxis**, **menuItemsYaxis**);  
 **kanjiUI**.setMovable(**false**);  
 **kanjiUI**.setVisible(**false**);  
  
 **mnemonicsUI** = **new** MnemonicsUI(**menuItemWindowWidth**, **menuItemWindowHeight**);  
 **mnemonicsUI**.setPosition(**menuItemsXaxis**, **menuItemsYaxis**);  
 **mnemonicsUI**.setMovable(**false**);  
 **mnemonicsUI**.setVisible(**false**);  
  
  
 Gdx.*app*.log(***TAG***, **"all\_health\_heart.size is: "** + **all\_health\_heart**.**size**);  
  
 **for** (**int** i = 0; i<**all\_health\_heart**.**size**; i++) {  
 **stage**.addActor(**all\_health\_heart**.get(i));  
 }  
  
 **battleUI** = **new** BattleUI();  
 **battleUI**.setFillParent(**true**);  
 **battleUI**.setVisible(**false**);  
 **battleUI**.setMovable(**false**);  
 *//removes all listeners including ones that handle focus* **battleUI**.clearListeners();  
  
 **stage**.addActor(**battleUI**);  
 **stage**.addActor(**menuButton**);  
 **stage**.addActor(**menuListUI**);  
 **stage**.addActor(**progressUI**);  
 **stage**.addActor(**inventoryUI**);  
 **stage**.addActor(**hiraganaUI**);  
 **stage**.addActor(**katakanaUI**);  
 **stage**.addActor(**kanjiUI**);  
 **stage**.addActor(**mnemonicsUI**);  
  
 *//Observers* ProfileManager.*getInstance*().addObserver(**this**);  
 **this**.addObserver(AudioManager.*getInstance*());  
 player.registerObserver(**this**);  
 **inventoryUI**.addObserver(**this**);  
 **battleUI**.getCurrentState().addObserver(**this**);  
  
 **menuButton**.addListener(**new** ClickListener() {  
 **public void** clicked (InputEvent event, **float** x, **float** y) {  
 *//menuListUI.setVisible(menuListUI.isVisible() ? false : true);  
 //MainGameScreen.setGameState(MainGameScreen.GameState.PAUSED);* **if** (**menuListUI**.isVisible()) {  
 **menuListUI**.setVisible(**false**);  
 multiplexer.addProcessor(player.getInputProcessor());  
 }  
 **else** {  
 **menuListUI**.setVisible(**true**);  
 multiplexer.removeProcessor(player.getInputProcessor());  
 }  
 **progressUI**.setVisible(**false**);  
 **inventoryUI**.setVisible(**false**);  
 **katakanaUI**.setVisible(**false**);  
 **hiraganaUI**.setVisible(**false**);  
 **mnemonicsUI**.setVisible(**false**);  
 **kanjiUI**.setVisible(**false**);  
 }  
 });  
  
 **progressButton** = **menuListUI**.getProgressButton();  
 **progressButton**.addListener(**new** ClickListener() {  
 **public void** clicked (InputEvent event, **float** x, **float** y) {  
 **inventoryUI**.setVisible(**false**);  
 **katakanaUI**.setVisible(**false**);  
 **hiraganaUI**.setVisible(**false**);  
 **kanjiUI**.setVisible(**false**);  
 **mnemonicsUI**.setVisible(**false**);  
 **progressUI**.setVisible(**progressUI**.isVisible()?**false**:**true**);  
 }  
 });  
  
 **inventoryButton** = **menuListUI**.getInventoryButton();  
 **inventoryButton**.addListener(**new** ClickListener() {  
 **public void** clicked (InputEvent event, **float** x, **float** y) {  
 **progressUI**.setVisible(**false**);  
 **katakanaUI**.setVisible(**false**);  
 **hiraganaUI**.setVisible(**false**);  
 **kanjiUI**.setVisible(**false**);  
 **mnemonicsUI**.setVisible(**false**);  
 **inventoryUI**.setVisible(**inventoryUI**.isVisible()?**false**:**true**);  
 }  
 });  
  
 **hiraganaButton** = **menuListUI**.getHiraganaButton();  
 **hiraganaButton**.addListener(**new** ClickListener() {  
 **public void** clicked (InputEvent event, **float** x, **float** y) {  
 **progressUI**.setVisible(**false**);  
 **inventoryUI**.setVisible(**false**);  
 **katakanaUI**.setVisible(**false**);  
 **kanjiUI**.setVisible(**false**);  
 **mnemonicsUI**.setVisible(**false**);  
 **hiraganaUI**.setVisible(**hiraganaUI**.isVisible()?**false**:**true**);  
  
 }  
 });  
  
 **katakanaButton** = **menuListUI**.getKatakanaButton();  
 **katakanaButton**.addListener(**new** ClickListener() {  
 **public void** clicked (InputEvent event, **float** x, **float** y) {  
 **progressUI**.setVisible(**false**);  
 **inventoryUI**.setVisible(**false**);  
 **hiraganaUI**.setVisible(**false**);  
 **kanjiUI**.setVisible(**false**);  
 **mnemonicsUI**.setVisible(**false**);  
 **katakanaUI**.setVisible(**katakanaUI**.isVisible()?**false**:**true**);  
 }  
 });  
  
 **kanjiButton** = **menuListUI**.getKanjiButton();  
 **kanjiButton**.addListener(**new** ClickListener() {  
 **public void** clicked (InputEvent event, **float** x, **float** y) {  
 **progressUI**.setVisible(**false**);  
 **inventoryUI**.setVisible(**false**);  
 **hiraganaUI**.setVisible(**false**);  
 **katakanaUI**.setVisible(**false**);  
 **mnemonicsUI**.setVisible(**false**);  
 **kanjiUI**.setVisible(**kanjiUI**.isVisible()?**false**:**true**);  
 }  
 });  
  
 **mnemonicsButton** = **menuListUI**.getMnemonicsButton();  
 **mnemonicsButton**.addListener(**new** ClickListener() {  
 **public void** clicked (InputEvent event, **float** x, **float** y) {  
 **progressUI**.setVisible(**false**);  
 **inventoryUI**.setVisible(**false**);  
 **hiraganaUI**.setVisible(**false**);  
 **katakanaUI**.setVisible(**false**);  
 **kanjiUI**.setVisible(**false**);  
 **mnemonicsUI**.setVisible(**mnemonicsUI**.isVisible()?**false**:**true**);  
 }  
 });  
  
 *//Music/Sound loading* notify(AudioObserver.AudioCommand.***MUSIC\_LOAD***, AudioObserver.AudioTypeEvent.***MUSIC\_BATTLE***);  
 notify(AudioObserver.AudioCommand.***SOUND\_LOAD***, AudioObserver.AudioTypeEvent.***SOUND\_PLAYER\_PAIN***);  
 notify(AudioObserver.AudioCommand.***SOUND\_LOAD***, AudioObserver.AudioTypeEvent.***SOUND\_EATING***);  
 notify(AudioObserver.AudioCommand.***SOUND\_LOAD***, AudioObserver.AudioTypeEvent.***SOUND\_DRINKING***);  
 notify(AudioObserver.AudioCommand.***SOUND\_LOAD***, AudioObserver.AudioTypeEvent.***SOUND\_SUCCESS***);  
 notify(AudioObserver.AudioCommand.***SOUND\_LOAD***, AudioObserver.AudioTypeEvent.***SOUND\_LVL\_UP***);  
 notify(AudioObserver.AudioCommand.***SOUND\_LOAD***, AudioObserver.AudioTypeEvent.***SOUND\_LVL\_DOWN***);  
 }  
  
 @Override  
 **public void** onNotify(ProfileManager profileManager, ProfileEvent event) {  
 **switch**(event){  
 **case *PROFILE\_LOADED***:  
 **boolean** firstTime = profileManager.getIsNewProfile();  
 **if**(firstTime){  
  
 Gdx.*app*.log(***TAG***, **"PROFILE LOADED First Time"**);  
 **maxNumberOfHearts** = 5;  
 **numberOfHearts** = 3;  
 showHearts();  
  
 InventoryUI.*clearInventoryItems*(**inventoryUI**.getInventorySlotTable());  
  
 *//add default items if first time* Array<ItemNameID> items = **player**.getEntityConfig().getInventory();  
 Array<InventoryItemLocation> itemLocations = **new** Array<InventoryItemLocation>();  
 **for**( **int** i = 0; i < items.**size**; i++){  
 itemLocations.add(**new** InventoryItemLocation(i, items.get(i).toString()));  
 }  
 InventoryUI.*populateInventory*(**inventoryUI**.getInventorySlotTable(), itemLocations);  
 profileManager.setProperty(**"playerInventory"**, InventoryUI.*getInventory*(**inventoryUI**.getInventorySlotTable()));  
  
 } **else** {  
 **int** hpMaxVal = profileManager.getProperty(**"currentPlayerHPMax"**, Integer.**class**);  
 **int** hpVal = profileManager.getProperty(**"currentPlayerHP"**, Integer.**class**);  
  
 **maxNumberOfHearts** = hpMaxVal;  
 **numberOfHearts** = hpVal;  
 showHearts();  
  
  
 ArrayList<KanaLetter> kanaLettersList = KanaLettersFactory.*getInstance*().getKanaLettersList();  
 ArrayList<KanjiLetter> kanjiLettersList = KanjiLettersFactory.*getInstance*().getKanjiLettersList();  
 Array<Integer> hiraganaProgress = profileManager.getProperty(**"hiraganaList"**, Array.**class**);  
 **for**(**int** i = 0; i <= hiraganaProgress.**size**-1; i++){  
 LetterLvlCounter.*adjustLvl*(kanaLettersList.get(i).getHiraganaEquivalent(),  
 hiraganaProgress.get(i));  
 }  
 Array<Integer> katakanaProgress = profileManager.getProperty(**"katakanaList"**, Array.**class**);  
 **for**(**int** i = 0; i <= katakanaProgress.**size**-1; i++){  
 LetterLvlCounter.*adjustLvl*(kanaLettersList.get(i).getKatakanaEquivalent(),  
 katakanaProgress.get(i));  
 }  
 Array<Integer> kanjiProgress = profileManager.getProperty(**"kanjiList"**, Array.**class**);  
 **for**(**int** i = 0; i <= kanjiProgress.**size**-1; i++){  
 LetterLvlCounter.*adjustLvl*(kanjiLettersList.get(i).getKanjiNameID(),  
 kanjiProgress.get(i));  
 }  
  
 **progressUI**.updateTable();  
  
 LetterLvlCounter.*setAllHiraganaMemorised*(profileManager.getProperty(**"allHiraganaMemorised"**, **boolean**.**class**));  
 LetterLvlCounter.*setAllKatakanaMemorised*(profileManager.getProperty(**"allKatakanaMemorised"**, **boolean**.**class**));  
 LetterLvlCounter.*setAllKanjiMemorised*(profileManager.getProperty(**"allKanjiMemorised"**, **boolean**.**class**));  
  
 Array<InventoryItemLocation> inventory = profileManager.getProperty(**"playerInventory"**, Array.**class**);  
 InventoryUI.*populateInventory*(**inventoryUI**.getInventorySlotTable(), inventory);  
 }  
  
 **break**;  
 **case *SAVING\_PROFILE***:  
 Gdx.*app*.log(***TAG***, **"PROFILE Saved"**);  
 profileManager.setProperty(**"playerInventory"**, **inventoryUI**.*getInventory*(**inventoryUI**.getInventorySlotTable()));  
 profileManager.setProperty(**"currentPlayerHPMax"**, **maxNumberOfHearts** );  
 profileManager.setProperty(**"currentPlayerHP"**, **numberOfHearts**);  
 profileManager.setProperty(**"hiraganaList"**, LetterLvlCounter.*getHiraganaList*());  
 profileManager.setProperty(**"katakanaList"**, LetterLvlCounter.*getKatakanaList*());  
 profileManager.setProperty(**"kanjiList"**, LetterLvlCounter.*getKanjiList*());  
 profileManager.setProperty(**"allHiraganaMemorised"**, LetterLvlCounter.*isAllHiraganaMemorised*());  
 profileManager.setProperty(**"allKatakanaMemorised"**, LetterLvlCounter.*isAllKatakanaMemorised*());  
 profileManager.setProperty(**"allKanjiMemorised"**, LetterLvlCounter.*isAllKanjiMemorised*());  
 **break**;  
 **case *CLEAR\_CURRENT\_PROFILE***:  
 profileManager.setProperty(**"playerInventory"**, **new** Array<InventoryItemLocation>());  
 profileManager.setProperty(**"currentPlayerHPMax"**, 5 );  
 profileManager.setProperty(**"currentPlayerHP"**, 3 );  
 profileManager.setProperty(**"hiraganaList"**, 1);  
 profileManager.setProperty(**"katakanaList"**, 1);  
 profileManager.setProperty(**"kanjiList"**, 1);  
 profileManager.setProperty(**"allHiraganaMemorised"**, **false**);  
 profileManager.setProperty(**"allKatakanaMemorised"**, **false**);  
 profileManager.setProperty(**"allKanjiMemorised"**, **false**);  
  
 **break**;  
 **default**:  
 **break**;  
 }  
 }  
  
 @Override  
 **public void** show() {  
 }  
  
 @Override  
 **public void** render(**float** delta) {  
 **stage**.act(delta);  
 **stage**.draw();  
  
 }  
  
 @Override *//****TODO speak about this in the report* public void** resize(**int** width, **int** height) {  
 **menuItemsXaxis** = 0;  
 **menuItemsYaxis** = height/40;  
 **menuItemWindowWidth** = width/1.4f;  
 **menuItemWindowHeight** = height/1.05f;  
  
 **for** (**int** i = 0; i<10; i++) {  
 **all\_health\_heart**.get(i).setPosition(**health\_heart**.getWidth() \* i, height - **health\_heart**.getHeight());  
 }  
  
 **menuButton**.setPosition(width/1.2f, height/12);  
  
 **menuListUI**.setPosition(width/1.27f, height/2);  
 **menuListUI**.updateSize(width/3.4f, height/1.4f);  
  
 **progressUI**.setPosition(**menuItemsXaxis**, **menuItemsYaxis**);  
 **progressUI**.updateSize(**menuItemWindowWidth**, **menuItemWindowHeight**);  
  
 **inventoryUI**.setPosition(**menuItemsXaxis**, **menuItemsYaxis**);  
 **inventoryUI**.updateSize(**menuItemWindowWidth**, **menuItemWindowHeight**);  
  
 **hiraganaUI**.setPosition(**menuItemsXaxis**, **menuItemsYaxis**);  
 **hiraganaUI**.updateSize(**menuItemWindowWidth**, **menuItemWindowHeight**);  
  
 **katakanaUI**.setPosition(**menuItemsXaxis**, **menuItemsYaxis**);  
 **katakanaUI**.updateSize(**menuItemWindowWidth**, **menuItemWindowHeight**);  
  
 **kanjiUI**.setPosition(**menuItemsXaxis**, **menuItemsYaxis**);  
 **kanjiUI**.updateSize(**menuItemWindowWidth**, **menuItemWindowHeight**);  
  
 **mnemonicsUI**.setPosition(**menuItemsXaxis**, **menuItemsYaxis**);  
 **mnemonicsUI**.updateSize(**menuItemWindowWidth**, **menuItemWindowHeight**);  
  
 **if**(width > 1280 && height > 720){  
 MainGameScreen.*setGameState*(MainGameScreen.GameState.***LOADING***);  
 }  
  
 **stage**.getViewport().update(width, height, **true**);  
 }  
  
 @Override  
 **public void** pause() {  
  
 }  
  
 @Override  
 **public void** resume() {  
  
 }  
  
 @Override  
 **public void** hide() {  
  
 }  
  
 @Override  
 **public void** dispose() {  
 **stage**.dispose();  
 **player**.dispose();  
 }  
  
 @Override  
 **public void** onNotify(String itemInfo, InventoryEvent event) {  
 **switch**(event){  
 **case *ITEM\_CONSUMED***:  
 String[] strings = itemInfo.split(Component.***MESSAGE\_TOKEN***);  
 **if**( strings.**length** != 2) **return**;  
  
 **int** type = Integer.*parseInt*(strings[0]);  
 **int** value = Integer.*parseInt*(strings[1]);  
  
 *//Gdx.app.log(TAG, "typeValue is: " + typeValue);* **if**( InventoryItem.*doesRestoreHP*(type) ){  
 notify(AudioObserver.AudioCommand.***SOUND\_PLAY\_ONCE***, AudioObserver.AudioTypeEvent.***SOUND\_DRINKING***);  
 **numberOfHearts** = MathUtils.*clamp*(**numberOfHearts** + value, 0, **maxNumberOfHearts**);  
 ProfileManager.*getInstance*().setProperty(**"currentPlayerHP"**, **numberOfHearts**);  
 showHearts();  
 }  
 **else if**( InventoryItem.*doesIncreaseMaxHP*(type) ){  
 notify(AudioObserver.AudioCommand.***SOUND\_PLAY\_ONCE***, AudioObserver.AudioTypeEvent.***SOUND\_EATING***);  
 **maxNumberOfHearts**++;  
 **numberOfHearts**++;  
 showHearts();  
 ProfileManager.*getInstance*().setProperty(**"currentPlayerHP"**, **numberOfHearts**);  
 ProfileManager.*getInstance*().setProperty(**"currentPlayerHPMax"**, **maxNumberOfHearts**);  
 }  
 **else if**(InventoryItem.*doesIncreaseHiraganaLvl*(type)){  
 notify(AudioObserver.AudioCommand.***SOUND\_PLAY\_ONCE***, AudioObserver.AudioTypeEvent.***SOUND\_LVL\_UP***);  
 LetterLvlCounter.*setAllHiraganaMemorised*(**true**);  
 **progressUI**.updateTable();  
 }  
 **else if**(InventoryItem.*doesDecreaseHiraganaLvl*(type)){  
 notify(AudioObserver.AudioCommand.***SOUND\_PLAY\_ONCE***, AudioObserver.AudioTypeEvent.***SOUND\_LVL\_DOWN***);  
 LetterLvlCounter.*setAllHiraganaMemorised*(**false**);  
 **progressUI**.updateTable();  
 }  
 **else if**(InventoryItem.*doesIncreaseKatakanaLvl*(type)){  
 notify(AudioObserver.AudioCommand.***SOUND\_PLAY\_ONCE***, AudioObserver.AudioTypeEvent.***SOUND\_LVL\_UP***);  
  
 LetterLvlCounter.*setAllKatakanaMemorised*(**true**);  
 **progressUI**.updateTable();  
 }  
 **else if**(InventoryItem.*doesDecreaseKatakanaLvl*(type)){  
 notify(AudioObserver.AudioCommand.***SOUND\_PLAY\_ONCE***, AudioObserver.AudioTypeEvent.***SOUND\_LVL\_DOWN***);  
 LetterLvlCounter.*setAllKatakanaMemorised*(**false**);  
 **progressUI**.updateTable();  
 }  
  
  
 **break**;  
 **default**:  
 **break**;  
 }  
 }  
  
 @Override  
 **public void** onNotify(String value, ComponentEvent event) {  
 **switch**(event) {  
 **case *ENEMY\_SPAWN\_LOCATION\_CHANGED***:  
 String enemyZoneID = value;  
 Gdx.*app*.debug(***TAG***, **"ENEMY\_SPAWN\_LOCATION\_CHANGED "** + enemyZoneID);  
 **battleUI**.battleZoneTriggered(Integer.*parseInt*(enemyZoneID));  
 **break**;  
 **case *PLAYER\_HAS\_MOVED***:  
 *//Gdx.app.debug(TAG, "PLAYER\_HAS\_MOVED ");* **if** (**battleUI**.isBattleReady()) {  
 MainGameScreen.*setGameState*(MainGameScreen.GameState.***SAVING***);  
 *//mapManager.disableCurrentmapMusic();  
 //notify(AudioObserver.AudioCommand.MUSIC\_PLAY\_LOOP, AudioObserver.AudioTypeEvent.MUSIC\_BATTLE);* **battleUI**.toBack();  
 **battleUI**.setVisible(**true**);  
 }  
 **break**;  
 **default**:  
 **break**;  
 }  
 }  
  
 @Override  
 **public void** onNotify(String answeredLetter, BattleEvent event) {  
 **int** hpVal;  
 **switch** (event) {  
 **case *LETTER\_ANSWERED\_CORRECTLY***:  
 notify(AudioObserver.AudioCommand.***SOUND\_PLAY\_ONCE***, AudioObserver.AudioTypeEvent.***SOUND\_SUCCESS***);  
 LetterLvlCounter.*increaseLvl*(answeredLetter, 1);  
 **progressUI**.updateTable();  
  
 MainGameScreen.*setGameState*(MainGameScreen.GameState.***RUNNING***);  
 *//notify(AudioObserver.AudioCommand.MUSIC\_STOP, AudioObserver.AudioTypeEvent.MUSIC\_BATTLE);  
 //mapManager.enableCurrentmapMusic();* **battleUI**.setVisible(**false**);  
 **break**;  
 **case *LETTER\_ANSWERED\_INCORRECTLY***:  
 notify(AudioObserver.AudioCommand.***SOUND\_PLAY\_ONCE***, AudioObserver.AudioTypeEvent.***SOUND\_PLAYER\_PAIN***);  
 LetterLvlCounter.*decreaseLvl*(answeredLetter, 1);  
 **progressUI**.updateTable();  
 **numberOfHearts**--;  
 showHearts();  
 ProfileManager.*getInstance*().setProperty(**"currentPlayerHP"**, **numberOfHearts**);  
 **if**( **numberOfHearts** <= 0 ){  
 *//notify(AudioObserver.AudioCommand.MUSIC\_STOP, AudioObserver.AudioTypeEvent.MUSIC\_BATTLE);* **battleUI**.setVisible(**false**);  
 MainGameScreen.*setGameState*(MainGameScreen.GameState.***GAME\_OVER***);  
 }  
 **break**;  
 **case *PLAYER\_RUNNING***:  
 MainGameScreen.*setGameState*(MainGameScreen.GameState.***RUNNING***);  
 *//mapManager.enableCurrentmapMusic();* **battleUI**.setVisible(**false**);  
 **break**;  
 **case *PLAYER\_HIT\_DAMAGE***:  
 notify(AudioObserver.AudioCommand.***SOUND\_PLAY\_ONCE***, AudioObserver.AudioTypeEvent.***SOUND\_PLAYER\_PAIN***);  
 **numberOfHearts**--;  
 showHearts();  
 ProfileManager.*getInstance*().setProperty(**"currentPlayerHP"**, **numberOfHearts**);  
 **if**( **numberOfHearts** <= 0 ){  
 **battleUI**.setVisible(**false**);  
 MainGameScreen.*setGameState*(MainGameScreen.GameState.***GAME\_OVER***);  
 }  
 **break**;  
 **default**:  
 **break**;  
 }  
 }  
  
 **public void** showHearts(){  
 **for** (**int** i = 0; i<**all\_health\_heart**.**size**; i++) {  
 **all\_health\_heart**.get(i).setVisible(**false**);  
 }  
 **for** (**int** j = 0; j < **numberOfHearts**; j++) {  
 **all\_health\_heart**.get(j).setVisible(**true**);  
 }  
 }  
  
 **public** Stage getStage() {  
 **return stage**;  
 }  
  
 @Override  
 **public void** addObserver(AudioObserver audioObserver) {  
 **observers**.add(audioObserver);  
 }  
  
 @Override  
 **public void** removeObserver(AudioObserver audioObserver) {  
 **observers**.removeValue(audioObserver, **true**);  
 }  
  
 @Override  
 **public void** removeAllObservers() {  
 **observers**.removeAll(**observers**, **true**);  
 }  
  
 @Override  
 **public void** notify(AudioObserver.AudioCommand command, AudioObserver.AudioTypeEvent event) {  
 **for**(AudioObserver observer: **observers**){  
 observer.onNotify(command, event);  
 }  
 }  
}

**package** com.mygdx.game.gui;  
  
**import** com.badlogic.gdx.scenes.scene2d.ui.Image;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Label;  
**import** com.badlogic.gdx.scenes.scene2d.ui.ScrollPane;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Table;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Window;  
**import** com.badlogic.gdx.utils.Align;  
**import** com.mygdx.game.japanese.KanaLetter;  
**import** com.mygdx.game.japanese.KanaLettersFactory;  
**import** com.mygdx.game.japanese.KanjiLetter;  
**import** com.mygdx.game.japanese.KanjiLettersFactory;  
**import** com.mygdx.game.japanese.LetterLvlCounter;  
**import** com.mygdx.game.tools.Utility;  
  
**import** java.util.ArrayList;  
  
**public class** ProgressUI **extends** Window{  
  
 **private final static** String ***TAG*** = ProgressUI.**class**.getSimpleName();  
  
 **private** Label **text**;  
 **private int tempInt**;  
 **private** String **tempString**;  
  
 **private** Table **table**;  
 **private** Table **hiraganaTable**;  
 **private** Table **katakanaTable**;  
 **private** Table **kanjiTable**;  
  
 **private** KanaLetter **kanaLetter**;  
 **private** KanjiLetter **kanjiLetter**;  
 **private** ArrayList<KanaLetter> **kanaLettersList**;  
 **private** ArrayList<KanjiLetter> **kanjiLettersList**;  
  
 **private** Image **equivalent**;  
 **private float menuItemWindowWidth**;  
 **private float menuItemWindowHeight**;  
  
 **public** ProgressUI(**float** width, **float** height){  
 **super**(**"Progress"**, Utility.*GUI\_SKINS*);  
 **this**.getTitleLabel().setAlignment(Align.***center***);  
  
 **menuItemWindowWidth** = width;  
 **menuItemWindowHeight** = height;  
  
 **this**.pad(**this**.getPadTop() + **menuItemWindowHeight** / 15, 10,  
 **menuItemWindowHeight** / 75, 10);  
  
  
 **table** = **new** Table();  
 **hiraganaTable** = **new** Table();  
 **katakanaTable** = **new** Table();  
 **kanjiTable** = **new** Table();  
  
 **kanaLettersList** = KanaLettersFactory.*getInstance*().getKanaLettersList();  
 **kanjiLettersList** = KanjiLettersFactory.*getInstance*().getKanjiLettersList();  
  
 **this**.setSize(**menuItemWindowWidth**, **menuItemWindowHeight**);  
 *//this.debug();* }  
  
 **public void** updateSize(**float** width, **float** height){  
 **menuItemWindowWidth** = width;  
 **menuItemWindowHeight** = height;  
  
 **this**.setSize(**menuItemWindowWidth**, **menuItemWindowHeight**);  
 }  
  
 *//****TODO speak about this* public void** updateTable(){  
 **hiraganaTable**.clear();  
 **katakanaTable**.clear();  
 **kanjiTable**.clear();  
 **table**.clear();  
 **this**.clear();  
  
 **text** = **new** Label(**"Hiragana"**, Utility.*GUI\_SKINS*);  
 **table**.add(**text**).left();  
 **table**.row();  
  
 **for**(**int** i = 0; i < **kanaLettersList**.size(); i++) {  
  
 **if**(i % 4 == 0 ){  
 **hiraganaTable**.row();  
 }  
  
 **int** counter = 0;  
 **kanaLetter** = **kanaLettersList**.get(i);  
 **tempString** = **kanaLetter**.getHiraganaEquivalent();  
 **tempInt** = LetterLvlCounter.*getHiraganaLvlTable*().get(**tempString**);  
 **equivalent** = **new** Image(Utility.*SMALL\_HIRAGANA\_TEXTUREATLAS*.findRegion(**tempString**));  
  
 **if**(LetterLvlCounter.*isAllHiraganaMemorised*()) {  
 **tempString** = **"memorised"**;  
 **text** = **new** Label(**tempString**, Utility.*GUI\_SKINS*, **"progress\_list\_text"**);  
 }  
 **else** {  
 counter += **tempInt**;  
 **if** (counter >= 428) {  
 LetterLvlCounter.*setAllHiraganaMemorised*(**true**);  
 }  
 **if** (**tempInt** >= 3) {  
 **tempString** = **"memorised"**;  
 **text** = **new** Label(**tempString** + **""**, Utility.*GUI\_SKINS*, **"progress\_list\_text"**);  
 } **else** {  
 **tempString** = **"Lvl: "** + LetterLvlCounter.*getHiraganaLvlTable*().get(**tempString**);  
 **text** = **new** Label(**tempString**, Utility.*GUI\_SKINS*, **"list\_text"**);  
 }  
 *//Gdx.app.debug(TAG, "hiraganaLvlCounter is " + hiraganaLvlCounter);* }  
  
 **hiraganaTable**.add(**equivalent**).left();  
 **hiraganaTable**.add(**text**).left().padRight(**this**.getWidth()/75);  
 }  
  
 **table**.add(**hiraganaTable**);  
 **table**.row();  
  
 **text** = **new** Label(**"Katakana"**, Utility.*GUI\_SKINS*);  
 **table**.add(**text**).left();  
 **table**.row();  
  
 **for**(**int** i = 0; i < **kanaLettersList**.size(); i++) {  
  
 **if**(i % 4 == 0 ){  
 **katakanaTable**.row();  
 }  
  
 **int** counter = 0;  
 **kanaLetter** = **kanaLettersList**.get(i);  
 **tempString** = **kanaLetter**.getKatakanaEquivalent();  
 **tempInt** = LetterLvlCounter.*getKatakanaLvlTable*().get(**tempString**);  
 **equivalent** = **new** Image(Utility.*SMALL\_KATAKANA\_TEXTUREATLAS*.findRegion(**tempString**));  
  
 **if**(LetterLvlCounter.*isAllKatakanaMemorised*()) {  
 **tempString** = **"memorised"**;  
 **text** = **new** Label(**tempString**, Utility.*GUI\_SKINS*, **"progress\_list\_text"**);  
 }  
 **else** {  
 counter += **tempInt**;  
 **if** (counter >= 428) {  
 LetterLvlCounter.*setAllKatakanaMemorised*(**true**);  
 }  
 **if** (**tempInt** >= 3) {  
 **tempString** = **"memorised"**;  
 **text** = **new** Label(**tempString** + **""**, Utility.*GUI\_SKINS*, **"progress\_list\_text"**);  
 } **else** {  
 **tempString** = **"Lvl: "** + LetterLvlCounter.*getKatakanaLvlTable*().get(**tempString**);  
 **text** = **new** Label(**tempString**, Utility.*GUI\_SKINS*, **"list\_text"**);  
 }  
 }  
  
 **katakanaTable**.add(**equivalent**).left();  
 **katakanaTable**.add(**text**).left().padRight(**this**.getWidth()/75);  
 }  
  
 **table**.add(**katakanaTable**);  
 **table**.row();  
  
 **text** = **new** Label(**"Kanji"**, Utility.*GUI\_SKINS*);  
 **table**.add(**text**).left();  
 **table**.row();  
  
 **for**(**int** i = 0; i < **kanjiLettersList**.size()-1; i++) {  
  
 **if**(i % 4 == 0 ){  
 **kanjiTable**.row();  
 }  
  
 **int** counter = 0;  
 **kanjiLetter** = **kanjiLettersList**.get(i);  
 **tempString** = **kanjiLetter**.getKanjiNameID();  
 **tempInt** = LetterLvlCounter.*getKanjiLvlTable*().get(**tempString**);  
 **equivalent** = **new** Image(Utility.*SMALL\_KANJI\_TEXTUREATLAS*.findRegion(**tempString**));  
  
 **if**(LetterLvlCounter.*isAllKanjiMemorised*()) {  
 **tempString** = **"memorised"**;  
 **text** = **new** Label(**tempString**, Utility.*GUI\_SKINS*, **"progress\_list\_text"**);  
 }  
 **else** {  
 counter += **tempInt**;  
 **if** (counter >= 175) {  
 LetterLvlCounter.*setAllKanjiMemorised*(**true**);  
 }  
 **if** (**tempInt** >= 5) {  
 **tempString** = **"memorised"**;  
 **text** = **new** Label(**tempString**, Utility.*GUI\_SKINS*, **"progress\_list\_text"**);  
 } **else** {  
 **tempString** = **"Lvl: "** + LetterLvlCounter.*getKanjiLvlTable*().get(**tempString**);  
 **text** = **new** Label(**tempString**, Utility.*GUI\_SKINS*, **"list\_text"**);  
 }  
 }  
  
 **kanjiTable**.add(**equivalent**).left();  
 **kanjiTable**.add(**text**).left().padRight(**this**.getWidth()/75);  
 }  
  
 **table**.add(**kanjiTable**);  
  
 ScrollPane scrollPane = **new** ScrollPane(**table**);  
 **this**.add(scrollPane).fill().expand();  
 **this**.setSize(**menuItemWindowWidth**, **menuItemWindowHeight**);  
 }  
  
  
}

## src.com.mygdx.game.inventory

**package** com.mygdx.game.inventory;  
  
**import** com.badlogic.gdx.graphics.g2d.TextureRegion;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Image;  
  
**public class** InventoryItem **extends** Image {  
  
 **public enum** ItemAttribute{  
 ***CONSUMABLE***(1),  
 ***STACKABLE***(2);  
  
 **private int attribute**;  
  
 ItemAttribute(**int** attribute){  
 **this**.**attribute** = attribute;  
 }  
  
 **public int** getValue(){  
 **return attribute**;  
 }  
  
 }  
  
 **public enum** ItemUseType{  
 ***ITEM\_RESTORE\_HEALTH***(1),  
 ***ITEM\_INCREASE\_MAX\_HP***(2),  
 ***ITEM\_INCREASE\_HIRAGANA\_LVL***(4),  
 ***ITEM\_DECREASE\_HIRAGANA\_LVL***(8),  
 ***ITEM\_INCREASE\_KATAKANA\_LVL***(16),  
 ***ITEM\_DECREASE\_KATAKANA\_LVL***(32);  
  
  
 **private int itemUseType**;  
  
 ItemUseType(**int** itemUseType){  
 **this**.**itemUseType** = itemUseType;  
 }  
  
 **public int** getValue(){  
 **return itemUseType**;  
 }  
 }  
  
 **public enum** ItemNameID {  
 ***ARROW\_DOWN***,***ARROW\_DOWN\_SMALL***,***ARROW\_UP***,  
 ***ARROW\_UP\_SMALL***,***DIAMOND***,***EGG***,***EMERALD***,  
 ***GOLDEN\_SKULL***,***H\_RUNESTONE***,***NEGATIVE\_H\_RUNESTONE***,***HEALTH\_HEART***,  
 ***HEART***,***K\_RUNESTONE***,***NEGATIVE\_K\_RUNESTONE***,***LARGE\_POTION\_BLUE***,  
 ***MEDIUM\_POTION\_BLUE***,***POTION\_BLUE***,  
 ***PYRAMID***, ***NONE***;  
 }  
  
 **private int itemAttributes**;  
 **private int itemUseType**;  
 **private** ItemNameID **itemNameID**;  
 **private** String **itemShortDescription**;  
 **private int itemUseValue**;  
  
 **public** InventoryItem(TextureRegion textureRegion, **int** itemAttributes, ItemNameID itemNameID, **int** itemUseType, **int** itemUseValue){  
 **super**(textureRegion);  
  
 **this**.**itemNameID** = itemNameID;  
 **this**.**itemAttributes** = itemAttributes;  
 **this**.**itemUseType** = itemUseType;  
 **this**.**itemUseValue** = itemUseValue;  
 }  
  
 **public** InventoryItem(){  
 **super**();  
 }  
  
 **public** InventoryItem(InventoryItem inventoryItem){  
 **super**();  
 **this**.**itemNameID** = inventoryItem.getItemNameID();  
 **this**.**itemAttributes** = inventoryItem.getItemAttributes();  
 **this**.**itemUseType** = inventoryItem.getItemUseType();  
 **this**.**itemShortDescription** = inventoryItem.getItemShortDescription();  
 **this**.**itemUseValue** = inventoryItem.getItemUseValue();  
 }  
  
 **public boolean** isConsumable(){  
 **return** ((**itemAttributes** & ItemAttribute.***CONSUMABLE***.getValue()) == ItemAttribute.***CONSUMABLE***.getValue());  
 }  
  
 **public static boolean** doesRestoreHP(**int** itemUseType){  
 **return** ((itemUseType & ItemUseType.***ITEM\_RESTORE\_HEALTH***.getValue()) == ItemUseType.***ITEM\_RESTORE\_HEALTH***.getValue());  
 }  
  
 **public static boolean** doesIncreaseMaxHP(**int** itemUseType){  
 **return** ((itemUseType & ItemUseType.***ITEM\_INCREASE\_MAX\_HP***.getValue()) == ItemUseType.***ITEM\_INCREASE\_MAX\_HP***.getValue());  
 }  
 **public static boolean** doesIncreaseHiraganaLvl(**int** itemUseType){  
 **return** ((itemUseType & ItemUseType.***ITEM\_INCREASE\_HIRAGANA\_LVL***.getValue()) == ItemUseType.***ITEM\_INCREASE\_HIRAGANA\_LVL***.getValue());  
 }  
 **public static boolean** doesDecreaseHiraganaLvl(**int** itemUseType){  
 **return** ((itemUseType & ItemUseType.***ITEM\_DECREASE\_HIRAGANA\_LVL***.getValue()) == ItemUseType.***ITEM\_DECREASE\_HIRAGANA\_LVL***.getValue());  
 }  
 **public static boolean** doesIncreaseKatakanaLvl(**int** itemUseType){  
 **return** ((itemUseType & ItemUseType.***ITEM\_INCREASE\_KATAKANA\_LVL***.getValue()) == ItemUseType.***ITEM\_INCREASE\_KATAKANA\_LVL***.getValue());  
 }  
 **public static boolean** doesDecreaseKatakanaLvl(**int** itemUseType){  
 **return** ((itemUseType & ItemUseType.***ITEM\_DECREASE\_KATAKANA\_LVL***.getValue()) == ItemUseType.***ITEM\_DECREASE\_KATAKANA\_LVL***.getValue());  
 }  
  
 **public boolean** isSameItem(InventoryItem candidateInventoryItem){  
 **return itemNameID** == candidateInventoryItem.getItemNameID();  
 }  
  
 **public int** getItemUseValue() {  
 **return itemUseValue**;  
 }  
  
 **public void** setItemUseValue(**int** itemUseValue) {  
 **this**.**itemUseValue** = itemUseValue;  
 }  
  
 **public int** getItemAttributes() {  
 **return itemAttributes**;  
 }  
  
 **public void** setItemAttributes(**int** itemAttributes) {  
 **this**.**itemAttributes** = itemAttributes;  
 }  
  
 **public** ItemNameID getItemNameID() {  
 **return itemNameID**;  
 }  
  
 **public void** setItemNameID(ItemNameID itemNameID) {  
 **this**.**itemNameID** = itemNameID;  
 }  
  
 **public int** getItemUseType() {  
 **return itemUseType**;  
 }  
  
 **public void** setItemUseType(**int** itemUseType) {  
 **this**.**itemUseType** = itemUseType;  
 }  
  
 **public** String getItemShortDescription() {  
 **return itemShortDescription**;  
 }  
  
 **public void** setItemShortDescription(String itemShortDescription) {  
 **this**.**itemShortDescription** = itemShortDescription;  
 }  
  
}

**package** com.mygdx.game.inventory;  
  
  
**import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.scenes.scene2d.utils.TextureRegionDrawable;  
**import** com.badlogic.gdx.utils.Json;  
**import** com.badlogic.gdx.utils.JsonValue;  
**import** com.badlogic.gdx.utils.Scaling;  
**import** java.util.ArrayList;  
**import** java.util.Hashtable;  
**import** com.mygdx.game.tools.Utility;  
**import** com.mygdx.game.inventory.InventoryItem.ItemNameID;  
  
**public class** InventoryItemFactory {  
  
 **private** Json **json** = **new** Json();  
 **private final** String **INVENTORY\_ITEM** = **"json\_scripts/inventory\_items.json"**;  
 **private static** InventoryItemFactory *instance* = **null**;  
 **private** Hashtable<ItemNameID,InventoryItem> **inventoryItemList**;  
  
 **public static** InventoryItemFactory getInstance() {  
 **if** (*instance* == **null**) {  
 *instance* = **new** InventoryItemFactory();  
 }  
  
 **return** *instance*;  
 }  
  
 **private** InventoryItemFactory(){  
 ArrayList<JsonValue> list = **json**.fromJson(ArrayList.**class**, Gdx.*files*.internal(**INVENTORY\_ITEM**));  
 **inventoryItemList** = **new** Hashtable<ItemNameID, InventoryItem>();  
  
 **for** (JsonValue jsonVal : list) {  
 InventoryItem inventoryItem = **json**.readValue(InventoryItem.**class**, jsonVal);  
 **inventoryItemList**.put(inventoryItem.getItemNameID(), inventoryItem);  
 }  
 }  
  
 **public** InventoryItem getInventoryItem(ItemNameID inventoryItemName){  
 InventoryItem item = **new** InventoryItem(**inventoryItemList**.get(inventoryItemName));  
 item.setDrawable(**new** TextureRegionDrawable(Utility.*ITEMS\_TEXTUREATLAS*.findRegion(item.getItemNameID().toString())));  
 item.setScaling(Scaling.***none***);  
 **return** item;  
 }  
  
}

**package** com.mygdx.game.inventory;  
  
**public class** InventoryItemLocation {  
 **private int locationIndex**;  
 **private** String **itemNameAtLocation**;  
  
 **public** InventoryItemLocation(){  
 }  
  
 **public** InventoryItemLocation( **int** locationIndex, String itemNameAtLocation){  
 **this**.**locationIndex** = locationIndex;  
 **this**.**itemNameAtLocation** = itemNameAtLocation;  
 }  
  
 **public** String getItemNameAtLocation() {  
 **return itemNameAtLocation**;  
 }  
  
 **public void** setItemNameAtLocation(String itemNameAtLocation) {  
 **this**.**itemNameAtLocation** = itemNameAtLocation;  
 }  
  
 **public int** getLocationIndex() {  
 **return locationIndex**;  
 }  
  
 **public void** setLocationIndex(**int** locationIndex) {  
 **this**.**locationIndex** = locationIndex;  
 }  
}

**package** com.mygdx.game.inventory;  
  
**import** com.badlogic.gdx.graphics.g2d.NinePatch;  
**import** com.badlogic.gdx.scenes.scene2d.Actor;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Image;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Label;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Stack;  
**import** com.badlogic.gdx.utils.Array;  
**import** com.badlogic.gdx.utils.SnapshotArray;  
**import** com.mygdx.game.tools.Utility;  
  
**public class** InventorySlot **extends** Stack {  
  
 *//All slots have this default image* **private** Stack **defaultBackground**;  
 **private** Label **numItemsLabel**;  
  
 **public** InventorySlot(){  
 **defaultBackground** = **new** Stack();  
 Image image = **new** Image(**new** NinePatch(Utility.*GUI\_TEXTUREATLAS*.createPatch(**"dialog"**)));  
  
 **numItemsLabel** = **new** Label(**""**, Utility.*GUI\_SKINS*);  
  
 **defaultBackground**.add(image);  
  
 **defaultBackground**.setName(**"background"**);  
  
 **this**.add(**defaultBackground**);  
 **this**.add(**numItemsLabel**);  
 }  
  
 @Override  
 **public void** add(Actor actor) {  
 **super**.add(actor);  
 }  
  
 **public void** remove(Actor actor) {  
 **super**.removeActor(actor);  
 }  
  
 **public void** add(Array<Actor> array) {  
 **for**( Actor actor : array){  
 **super**.add(actor);  
 }  
 }  
  
 **public void** clearAllInventoryItems() {  
 **if**( hasItem() ){  
 SnapshotArray<Actor> arrayChildren = **this**.getChildren();  
 **int** numInventoryItems = getNumItems();  
  
 **for**(**int** i = 0; i < numInventoryItems; i++) {  
 arrayChildren.pop();  
 }  
 }  
 }  
  
 **public boolean** hasItem(){  
 **if**( hasChildren() ){  
 SnapshotArray<Actor> items = **this**.getChildren();  
 **if**( items.**size** > 2 ){  
 **return true**;  
 }  
 }  
 **return false**;  
 }  
  
 **public int** getNumItems(){  
 **if**( hasChildren() ){  
 SnapshotArray<Actor> items = **this**.getChildren();  
 **return** items.**size** - 2;  
 }  
 **return** 0;  
 }  
  
 **public** InventoryItem getTopInventoryItem(){  
 InventoryItem actor = **null**;  
 **if**( hasChildren() ){  
 SnapshotArray<Actor> items = **this**.getChildren();  
 **if**( items.**size** > 2 ){  
 actor = (InventoryItem) items.peek();  
 }  
 }  
 **return** actor;  
 }  
  
}

## src.com.mygdx.game.japanese

**package** com.mygdx.game.japanese;  
  
**import** com.badlogic.gdx.scenes.scene2d.ui.Image;  
  
**public class** KanaLetter **extends** Image {  
  
 **private** String **hiraganaEquivalent**;  
 **private** String katakanaEquivalent;  
 **private** String **romajiEquivalent**;  
  
 **public** KanaLetter(){  
 **super**();  
 }  
  
 **public** KanaLetter(KanaLetter kanaLetter){  
 **super**();  
 **this**.**hiraganaEquivalent** = kanaLetter.getHiraganaEquivalent();  
 **this**.katakanaEquivalent = kanaLetter.getKatakanaEquivalent();  
 **this**.**romajiEquivalent** = kanaLetter.getRomajiEquivalent();  
 }  
  
 **public** String getHiraganaEquivalent() {  
 **return hiraganaEquivalent**;  
 }  
  
 **public void** setHiraganaNameID(String hiraganaNameID) {  
 **this**.**hiraganaEquivalent** = hiraganaNameID;  
 }  
  
 **public** String getKatakanaEquivalent() {  
 **return** katakanaEquivalent;  
 }  
  
 **public void** setKatakanaEquivalent(String katakanaEquivalent) {  
 **this**.katakanaEquivalent = katakanaEquivalent;  
 }  
  
 **public** String getRomajiEquivalent() {  
 **return romajiEquivalent**;  
 }  
  
 **public void** setRomajiEquivalent(String romajiEquivalent) {  
 **this**.**romajiEquivalent** = romajiEquivalent;  
 }  
  
  
}

**package** com.mygdx.game.japanese;  
  
**import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.scenes.scene2d.utils.TextureRegionDrawable;  
**import** com.badlogic.gdx.utils.Json;  
**import** com.badlogic.gdx.utils.JsonValue;  
**import** com.badlogic.gdx.utils.Scaling;  
**import** com.mygdx.game.tools.Utility;  
  
**import** java.util.ArrayList;  
**import** java.util.Hashtable;  
  
**public class** KanaLettersFactory {  
  
 **private** Json **json** = **new** Json();  
 **private final** String **kanaJsonFilePath** = **"json\_scripts/kana.json"**;  
 **private static** KanaLettersFactory *instance* = **null**;  
 **private** Hashtable<String, KanaLetter> **hiraganaLettersTable**;  
 **private** Hashtable<String, KanaLetter> **katakanaLettersTable**;  
 **private** ArrayList<KanaLetter> **kanaLettersList**;  
  
 **public static** KanaLettersFactory getInstance() {  
 **if** (*instance* == **null**) {  
 *instance* = **new** KanaLettersFactory();  
 }  
  
 **return** *instance*;  
 }  
  
 **private** KanaLettersFactory(){  
 ArrayList<JsonValue> list = **json**.fromJson(ArrayList.**class**, Gdx.*files*.internal(**kanaJsonFilePath**));  
 **hiraganaLettersTable** = **new** Hashtable<String , KanaLetter>();  
 **katakanaLettersTable** = **new** Hashtable<String , KanaLetter>();  
 **kanaLettersList** = **new** ArrayList<KanaLetter>();  
  
 **for** (JsonValue jsonVal : list) {  
 KanaLetter kanaLetter = **json**.readValue(KanaLetter.**class**, jsonVal);  
 **hiraganaLettersTable**.put(kanaLetter.getHiraganaEquivalent(), kanaLetter);  
 **katakanaLettersTable**.put(kanaLetter.getKatakanaEquivalent(), kanaLetter);  
 **kanaLettersList**.add(kanaLetter);  
 }  
 }  
  
 **public** KanaLetter getHiraganaLetter(String hiraganaNameID){  
 KanaLetter kanaLetter = **new** KanaLetter(**hiraganaLettersTable**.get(hiraganaNameID));  
 kanaLetter.setDrawable(**new** TextureRegionDrawable(Utility.*LARGE\_HIRAGANA\_TEXTUREATLAS*.  
 findRegion(kanaLetter.getHiraganaEquivalent())));  
 kanaLetter.setScaling(Scaling.***none***);  
 **return** kanaLetter;  
 }  
  
 **public** KanaLetter getKatakanaLetter(String katakanaNameID){  
 KanaLetter kanaLetter = **new** KanaLetter(**katakanaLettersTable**.get(katakanaNameID));  
 kanaLetter.setDrawable(**new** TextureRegionDrawable(Utility.*LARGE\_KATAKANA\_TEXTUREATLAS*.  
 findRegion(kanaLetter.getKatakanaEquivalent())));  
 kanaLetter.setScaling(Scaling.***none***);  
 **return** kanaLetter;  
 }  
  
 **public** ArrayList<KanaLetter> getKanaLettersList() {  
 **return kanaLettersList**;  
 }  
}

**package** com.mygdx.game.japanese;  
  
**import** com.badlogic.gdx.scenes.scene2d.ui.Image;  
**import** com.badlogic.gdx.utils.Array;  
**import** java.lang.String;  
  
**public class** KanjiLetter **extends** Image {  
  
 **private** String **kanjiNameID**;  
 **private** String **kanjiMeaning**;  
 **private** String **kanjiMnemonic**;  
 **private** String **hiraganaEquivalent**;  
  
 **public** KanjiLetter(){  
 **super**();  
 }  
  
 **public** KanjiLetter(KanjiLetter kanaLetter){  
 **super**();  
 **this**.**kanjiNameID** = kanaLetter.getKanjiNameID();  
 **this**.**kanjiMeaning** = kanaLetter.getKanjiMeaning();  
 **this**.**kanjiMnemonic** = kanaLetter.getKanjiMnemonic();  
 **this**.**hiraganaEquivalent** = kanaLetter.getHiraganaEquivalent();  
 }  
  
 **public** String getKanjiNameID() {  
 **return kanjiNameID**;  
 }  
  
 **public void** setKanjiNameID(String kanjiNameID) {  
 **this**.**kanjiNameID** = kanjiNameID;  
 }  
  
 **public** String getKanjiMeaning() {  
 **return kanjiMeaning**;  
 }  
  
 **public void** setKanjiMeaning(String kanjiMeaning) {  
 **this**.**kanjiMeaning** = kanjiMeaning;  
 }  
  
 **public** String getKanjiMnemonic() {  
 **return kanjiMnemonic**;  
 }  
  
 **public void** setKanjiMnemonic(String kanjiMnemonic) {  
 **this**.**kanjiMnemonic** = kanjiMnemonic;  
 }  
  
 **public** String getHiraganaEquivalent() {  
 **return hiraganaEquivalent**;  
 }  
  
 **public void** setHiraganaEquivalent(String hiraganaEquivalent) {  
 **this**.**hiraganaEquivalent** = hiraganaEquivalent;  
 }  
}

**package** com.mygdx.game.japanese;  
  
**import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.scenes.scene2d.utils.TextureRegionDrawable;  
**import** com.badlogic.gdx.utils.Json;  
**import** com.badlogic.gdx.utils.JsonValue;  
**import** com.badlogic.gdx.utils.Scaling;  
**import** com.mygdx.game.tools.Utility;  
  
**import** java.util.ArrayList;  
**import** java.util.Hashtable;  
  
**public class** KanjiLettersFactory {  
  
 **private** Json **json** = **new** Json();  
 **private final** String **kanjiJsonFilePath** = **"json\_scripts/kanji.json"**;  
 **private static** KanjiLettersFactory *instance* = **null**;  
 **private** Hashtable<String, KanjiLetter> **kanjiLettersTable**;  
 **private** ArrayList<KanjiLetter> **kanjiLettersList**;  
  
 **public static** KanjiLettersFactory getInstance() {  
 **if** (*instance* == **null**) {  
 *instance* = **new** KanjiLettersFactory();  
 }  
  
 **return** *instance*;  
 }  
  
 **private** KanjiLettersFactory(){  
 ArrayList<JsonValue> list = **json**.fromJson(ArrayList.**class**, Gdx.*files*.internal(**kanjiJsonFilePath**));  
 **kanjiLettersTable** = **new** Hashtable<String , KanjiLetter>();  
 **kanjiLettersList** = **new** ArrayList<KanjiLetter>();  
  
 **for** (JsonValue jsonVal : list) {  
 KanjiLetter kanjiLetter = **json**.readValue(KanjiLetter.**class**, jsonVal);  
 **kanjiLettersTable**.put(kanjiLetter.getKanjiNameID(), kanjiLetter);  
 **kanjiLettersList**.add(kanjiLetter);  
 }  
 }  
  
 **public** KanjiLetter getKanjiLetter(String kanjiNameID){  
 KanjiLetter kanjiLetter = **new** KanjiLetter(**kanjiLettersTable**.get(kanjiNameID));  
 **return** kanjiLetter;  
 }  
  
  
 **public** ArrayList<KanjiLetter> getKanjiLettersList() {  
 **return kanjiLettersList**;  
 }  
}

**package** com.mygdx.game.japanese;  
  
  
**import** com.badlogic.gdx.math.MathUtils;  
**import** com.badlogic.gdx.utils.Array;  
  
**import** java.util.Hashtable;  
  
*/\*\*  
 \* Created by firasAltayeb on 17/14/2117.  
 \*/***public class** LetterLvlCounter {  
  
 **public static int** *hiraganaALvl* = 1, *hiraganaKaLvl* = 1,  
 *hiraganaILvl* = 1, *hiraganaKiLvl* = 1,  
 *hiraganaULvl* = 1, *hiraganaKuLvl* = 1,  
 *hiraganaELvl* = 1, *hiraganaKeLvl* = 1,  
 *hiraganaOLvl* = 1, *hiraganaKoLvl* = 1,  
  
 *hiraganaSaLvl* = 1, *hiraganaTaLvl* = 1,  
 *hiraganaShiLvl* = 1, *hiraganaChiLvl* = 1,  
 *hiraganaSuLvl* = 1, *hiraganaTsuLvl* = 1,  
 *hiraganaSeLvl* = 1, *hiraganaTeLvl* = 1,  
 *hiraganaSoLvl* = 1, *hiraganaToLvl* = 1,  
  
 *hiraganaNaLvl* = 1, *hiraganaHaLvl* = 1,  
 *hiraganaNiLvl* = 1, *hiraganaHiLvl* = 1,  
 *hiraganaNuLvl* = 1, *hiraganaHuLvl* = 1,  
 *hiraganaNeLvl* = 1, *hiraganaHeLvl* = 1,  
 *hiraganaNoLvl* = 1, *hiraganaHoLvl* = 1,  
  
 *hiraganaMaLvl* = 1, *hiraganaRaLvl* = 1,  
 *hiraganaMiLvl* = 1, *hiraganaRiLvl* = 1,  
 *hiraganaMuLvl* = 1, *hiraganaRuLvl* = 1,  
 *hiraganaMeLvl* = 1, *hiraganaReLvl* = 1,  
 *hiraganaMoLvl* = 1, *hiraganaRoLvl* = 1,  
  
 *hiraganaNLvl* = 1,  
  
 *hiraganaGaLvl* = 1, *hiraganaZaLvl* = 1,  
 *hiraganaGiLvl* = 1, *hiraganaJiLvl* = 1,  
 *hiraganaGuLvl* = 1, *hiraganaZuLvl* = 1,  
 *hiraganaGeLvl* = 1, *hiraganaZeLvl* = 1,  
 *hiraganaGoLvl* = 1, *hiraganaZoLvl* = 1,  
  
 *hiraganaDaLvl* = 1, *hiraganaBaLvl* = 1,  
 *hiraganaDiLvl* = 1, *hiraganaBiLvl* = 1,  
 *hiraganaDuLvl* = 1, *hiraganaBuLvl* = 1,  
 *hiraganaDeLvl* = 1, *hiraganaBeLvl* = 1,  
 *hiraganaDoLvl* = 1, *hiraganaBoLvl* = 1,  
  
 *hiraganaPaLvl* = 1, *hiraganaYaLvl* = 1,  
 *hiraganaPiLvl* = 1, *hiraganaYuLvl* = 1,  
 *hiraganaPuLvl* = 1, *hiraganaYoLvl* = 1,  
 *hiraganaPeLvl* = 1, *hiraganaWaLvl* = 1,  
 *hiraganaPoLvl* = 1, *hiraganaWoLvl* = 1,  
  
 *hiraganaKyaLvl* = 1, *hiraganaShaLvl* = 1,  
 *hiraganaKyuLvl* = 1, *hiraganaShuLvl* = 1,  
 *hiraganaKyoLvl* = 1, *hiraganaShoLvl* = 1,  
  
 *hiraganaChaLvl* = 1, *hiraganaHyaLvl* = 1,  
 *hiraganaChuLvl* = 1, *hiraganaHyuLvl* = 1,  
 *hiraganaChoLvl* = 1, *hiraganaHyoLvl* = 1,  
  
 *hiraganaNyaLvl* = 1, *hiraganaMyaLvl* = 1,  
 *hiraganaNyuLvl* = 1, *hiraganaMyuLvl* = 1,  
 *hiraganaNyoLvl* = 1, *hiraganaMyoLvl* = 1,  
  
 *hiraganaRyaLvl* = 1, *hiraganaGyaLvl* = 1,  
 *hiraganaRyuLvl* = 1, *hiraganaGyuLvl* = 1,  
 *hiraganaRyoLvl* = 1, *hiraganaGyoLvl* = 1,  
  
 *hiraganaJyaLvl* = 1, *hiraganaDyaLvl* = 1,  
 *hiraganaJyuLvl* = 1, *hiraganaDyuLvl* = 1,  
 *hiraganaJyoLvl* = 1, *hiraganaDyoLvl* = 1,  
  
 *hiraganaByaLvl* = 1, *hiraganaPyaLvl* = 1,  
 *hiraganaByuLvl* = 1, *hiraganaPyuLvl* = 1,  
 *hiraganaByoLvl* = 1, *hiraganaPyoLvl* = 1,  
  
 *katakanaALvl* = 1, *katakanaKaLvl* = 1,  
 *katakanaILvl* = 1, *katakanaKiLvl* = 1,  
 *katakanaULvl* = 1, *katakanaKuLvl* = 1,  
 *katakanaELvl* = 1, *katakanaKeLvl* = 1,  
 *katakanaOLvl* = 1, *katakanaKoLvl* = 1,  
  
 *katakanaSaLvl* = 1, *katakanaTaLvl* = 1,  
 *katakanaShiLvl* = 1, *katakanaChiLvl* = 1,  
 *katakanaSuLvl* = 1, *katakanaTsuLvl* = 1,  
 *katakanaSeLvl* = 1, *katakanaTeLvl* = 1,  
 *katakanaSoLvl* = 1, *katakanaToLvl* = 1,  
  
 *katakanaNaLvl* = 1, *katakanaHaLvl* = 1,  
 *katakanaNiLvl* = 1, *katakanaHiLvl* = 1,  
 *katakanaNuLvl* = 1, *katakanaHuLvl* = 1,  
 *katakanaNeLvl* = 1, *katakanaHeLvl* = 1,  
 *katakanaNoLvl* = 1, *katakanaHoLvl* = 1,  
  
 *katakanaMaLvl* = 1, *katakanaRaLvl* = 1,  
 *katakanaMiLvl* = 1, *katakanaRiLvl* = 1,  
 *katakanaMuLvl* = 1, *katakanaRuLvl* = 1,  
 *katakanaMeLvl* = 1, *katakanaReLvl* = 1,  
 *katakanaMoLvl* = 1, *katakanaRoLvl* = 1,  
  
 *katakanaNLvl* = 1,  
  
 *katakanaGaLvl* = 1, *katakanaZaLvl* = 1,  
 *katakanaGiLvl* = 1, *katakanaJiLvl* = 1,  
 *katakanaGuLvl* = 1, *katakanaZuLvl* = 1,  
 *katakanaGeLvl* = 1, *katakanaZeLvl* = 1,  
 *katakanaGoLvl* = 1, *katakanaZoLvl* = 1,  
  
 *katakanaDaLvl* = 1, *katakanaBaLvl* = 1,  
 *katakanaDiLvl* = 1, *katakanaBiLvl* = 1,  
 *katakanaDuLvl* = 1, *katakanaBuLvl* = 1,  
 *katakanaDeLvl* = 1, *katakanaBeLvl* = 1,  
 *katakanaDoLvl* = 1, *katakanaBoLvl* = 1,  
  
 *katakanaPaLvl* = 1, *katakanaYaLvl* = 1,  
 *katakanaPiLvl* = 1, *katakanaYuLvl* = 1,  
 *katakanaPuLvl* = 1, *katakanaYoLvl* = 1,  
 *katakanaPeLvl* = 1, *katakanaWaLvl* = 1,  
 *katakanaPoLvl* = 1, *katakanaWoLvl* = 1,  
  
 *katakanaKyaLvl* = 1, *katakanaShaLvl* = 1,  
 *katakanaKyuLvl* = 1, *katakanaShuLvl* = 1,  
 *katakanaKyoLvl* = 1, *katakanaShoLvl* = 1,  
  
 *katakanaChaLvl* = 1, *katakanaHyaLvl* = 1,  
 *katakanaChuLvl* = 1, *katakanaHyuLvl* = 1,  
 *katakanaChoLvl* = 1, *katakanaHyoLvl* = 1,  
  
 *katakanaNyaLvl* = 1, *katakanaMyaLvl* = 1,  
 *katakanaNyuLvl* = 1, *katakanaMyuLvl* = 1,  
 *katakanaNyoLvl* = 1, *katakanaMyoLvl* = 1,  
  
 *katakanaRyaLvl* = 1, *katakanaGyaLvl* = 1,  
 *katakanaRyuLvl* = 1, *katakanaGyuLvl* = 1,  
 *katakanaRyoLvl* = 1, *katakanaGyoLvl* = 1,  
  
 *katakanaJyaLvl* = 1, *katakanaDyaLvl* = 1,  
 *katakanaJyuLvl* = 1, *katakanaDyuLvl* = 1,  
 *katakanaJyoLvl* = 1, *katakanaDyoLvl* = 1,  
  
 *katakanaByaLvl* = 1, *katakanaPyaLvl* = 1,  
 *katakanaByuLvl* = 1, *katakanaPyuLvl* = 1,  
 *katakanaByoLvl* = 1, *katakanaPyoLvl* = 1,  
  
 *kanjiHaLvl* = 1, *kanjiHanaLvl* = 1,  
 *kanjiKiLvl* = 1, *kanjiMoriLvl* = 1,  
 *kanjiMushiLvl* = 1, *kanjiIshiLvl* = 1,  
 *kanjiKazeLvl* = 1, *kanjiTsuchiLvl* = 1,  
 *kanjiUchiLvl* = 1, *kanjiYamaLvl* = 1,  
 *kanjiUmiLvl* = 1, *kanjiSakanaLvl* = 1,  
 *kanjiMizuLvl* = 1, *kanjiKaiLvl* = 1,  
 *kanjiIkeLvl* = 1, *kanjiYukiLvl* = 1,  
 *kanjiKooriLvl* = 1, *kanjiKawaLvl* = 1,  
 *kanjiHayashiLvl* = 1, *kanjiFuyuLvl* = 1,  
 *kanjiToLvl* = 1, *kanjiMonLvl* = 1,  
 *kanjiTeraLvl* = 1, *kanjiAnaLvl* = 1,  
 *kanjiSotoLvl* = 1, *kanjiKaLvl* = 1,  
 *kanjiHiLvl* = 1, *kanjiChiLvl* = 1,  
 *kanjiAkaLvl* = 1, *kanjiHikariLvl* = 1;  
  
  
 **private static boolean** *allHiraganaMemorised* = **false**;  
 **private static boolean** *allKatakanaMemorised* = **false**;  
 **private static boolean** *allKanjiMemorised* = **false**;  
 **private static** Hashtable<String, Integer> *hiraganaLvlTable*;  
 **private static** Hashtable<String, Integer> *katakanaLvlTable*;  
 **private static** Hashtable<String, Integer> *kanjiLvlTable*;  
 **private static** Array<Integer> *hiraganaList*;  
 **private static** Array<Integer> *katakanaList*;  
 **private static** Array<Integer> *kanjiList*;  
  
 **public static** Hashtable<String, Integer> getHiraganaLvlTable(){  
  
 *hiraganaLvlTable* = **new** Hashtable<String, Integer>();  
  
 *hiraganaLvlTable*.put(**"hiraganaA"**, *hiraganaALvl*);  
 *hiraganaLvlTable*.put(**"hiraganaI"**, *hiraganaILvl*);  
 *hiraganaLvlTable*.put(**"hiraganaU"**, *hiraganaULvl*);  
 *hiraganaLvlTable*.put(**"hiraganaE"**, hiraganaELvl);  
 hiraganaLvlTable.put(**"hiraganaO"**, hiraganaOLvl);  
 hiraganaLvlTable.put(**"hiraganaKa"**, hiraganaKaLvl);  
 hiraganaLvlTable.put(**"hiraganaKi"**, hiraganaKiLvl);  
 hiraganaLvlTable.put(**"hiraganaKu"**, hiraganaKuLvl);  
 hiraganaLvlTable.put(**"hiraganaKe"**, hiraganaKeLvl);  
 hiraganaLvlTable.put(**"hiraganaKo"**, hiraganaKoLvl);  
 hiraganaLvlTable.put(**"hiraganaSa"**, hiraganaSaLvl);  
 hiraganaLvlTable.put(**"hiraganaShi"**, hiraganaShiLvl);  
 hiraganaLvlTable.put(**"hiraganaSu"**, hiraganaSuLvl);  
 hiraganaLvlTable.put(**"hiraganaSe"**, hiraganaSeLvl);  
 hiraganaLvlTable.put(**"hiraganaSo"**, hiraganaSoLvl);  
 hiraganaLvlTable.put(**"hiraganaTa"**, hiraganaTaLvl);  
 hiraganaLvlTable.put(**"hiraganaChi"**, hiraganaChiLvl);  
 hiraganaLvlTable.put(**"hiraganaTsu"**, hiraganaTsuLvl);  
 hiraganaLvlTable.put(**"hiraganaTe"**, hiraganaTeLvl);  
 hiraganaLvlTable.put(**"hiraganaTo"**, hiraganaToLvl);  
 hiraganaLvlTable.put(**"hiraganaNa"**, hiraganaNaLvl);  
 hiraganaLvlTable.put(**"hiraganaNi"**, hiraganaNiLvl);  
 hiraganaLvlTable.put(**"hiraganaNu"**, hiraganaNuLvl);  
 hiraganaLvlTable.put(**"hiraganaNe"**, hiraganaNeLvl);  
 hiraganaLvlTable.put(**"hiraganaNo"**, hiraganaNoLvl);  
 hiraganaLvlTable.put(**"hiraganaHa"**, hiraganaHaLvl);  
 hiraganaLvlTable.put(**"hiraganaHi"**, hiraganaHiLvl);  
 hiraganaLvlTable.put(**"hiraganaHu"**, hiraganaHuLvl);  
 hiraganaLvlTable.put(**"hiraganaHe"**, hiraganaHeLvl);  
 hiraganaLvlTable.put(**"hiraganaHo"**, hiraganaHoLvl);  
 hiraganaLvlTable.put(**"hiraganaMa"**, hiraganaMaLvl);  
 hiraganaLvlTable.put(**"hiraganaMi"**, hiraganaMiLvl);  
 hiraganaLvlTable.put(**"hiraganaMu"**, hiraganaMuLvl);  
 hiraganaLvlTable.put(**"hiraganaMe"**, hiraganaMeLvl);  
 hiraganaLvlTable.put(**"hiraganaMo"**, hiraganaMoLvl);  
 hiraganaLvlTable.put(**"hiraganaRa"**, hiraganaRaLvl);  
 hiraganaLvlTable.put(**"hiraganaRi"**, hiraganaRiLvl);  
 hiraganaLvlTable.put(**"hiraganaRu"**, hiraganaRuLvl);  
 hiraganaLvlTable.put(**"hiraganaRe"**, hiraganaReLvl);  
 hiraganaLvlTable.put(**"hiraganaRo"**, hiraganaRoLvl);  
 hiraganaLvlTable.put(**"hiraganaYu"**, hiraganaYuLvl);  
 hiraganaLvlTable.put(**"hiraganaYo"**, hiraganaYoLvl);  
 hiraganaLvlTable.put(**"hiraganaWa"**, hiraganaWaLvl);  
 hiraganaLvlTable.put(**"hiraganaWo"**, hiraganaWoLvl);  
  
 hiraganaLvlTable.put(**"hiraganaN"**, hiraganaNLvl);  
  
 hiraganaLvlTable.put(**"hiraganaGa"**, hiraganaGaLvl);  
 hiraganaLvlTable.put(**"hiraganaGi"**, hiraganaGiLvl);  
 hiraganaLvlTable.put(**"hiraganaGu"**, hiraganaGuLvl);  
 hiraganaLvlTable.put(**"hiraganaGe"**, hiraganaGeLvl);  
 hiraganaLvlTable.put(**"hiraganaGo"**, hiraganaGoLvl);  
 hiraganaLvlTable.put(**"hiraganaZa"**, hiraganaZaLvl);  
 hiraganaLvlTable.put(**"hiraganaJi"**, hiraganaJiLvl);  
 hiraganaLvlTable.put(**"hiraganaZu"**, hiraganaZuLvl);  
 hiraganaLvlTable.put(**"hiraganaZe"**, hiraganaZeLvl);  
 hiraganaLvlTable.put(**"hiraganaZo"**, hiraganaZoLvl);  
 hiraganaLvlTable.put(**"hiraganaDa"**, hiraganaDaLvl);  
 hiraganaLvlTable.put(**"hiraganaDi"**, hiraganaDiLvl);  
 hiraganaLvlTable.put(**"hiraganaDu"**, hiraganaDuLvl);  
 hiraganaLvlTable.put(**"hiraganaDe"**, hiraganaDeLvl);  
 hiraganaLvlTable.put(**"hiraganaDo"**, hiraganaDoLvl);  
 hiraganaLvlTable.put(**"hiraganaBa"**, hiraganaBaLvl);  
 hiraganaLvlTable.put(**"hiraganaBi"**, hiraganaBiLvl);  
 hiraganaLvlTable.put(**"hiraganaBu"**, hiraganaBuLvl);  
 hiraganaLvlTable.put(**"hiraganaBe"**, hiraganaBeLvl);  
 hiraganaLvlTable.put(**"hiraganaBo"**, hiraganaBoLvl);  
 hiraganaLvlTable.put(**"hiraganaPa"**, hiraganaPaLvl);  
 hiraganaLvlTable.put(**"hiraganaPi"**, hiraganaPiLvl);  
 hiraganaLvlTable.put(**"hiraganaPu"**, hiraganaPuLvl);  
 hiraganaLvlTable.put(**"hiraganaPe"**, hiraganaPeLvl);  
 hiraganaLvlTable.put(**"hiraganaPo"**, hiraganaPoLvl);  
 hiraganaLvlTable.put(**"hiraganaYa"**, hiraganaYaLvl);  
 hiraganaLvlTable.put(**"hiraganaKya"**, hiraganaKyaLvl);  
 hiraganaLvlTable.put(**"hiraganaKyu"**, hiraganaKyuLvl);  
 hiraganaLvlTable.put(**"hiraganaKyo"**, hiraganaKyoLvl);  
 hiraganaLvlTable.put(**"hiraganaSha"**, hiraganaShaLvl);  
 hiraganaLvlTable.put(**"hiraganaShu"**, hiraganaShuLvl);  
 hiraganaLvlTable.put(**"hiraganaSho"**, hiraganaShoLvl);  
 hiraganaLvlTable.put(**"hiraganaCha"**, hiraganaChaLvl);  
 hiraganaLvlTable.put(**"hiraganaChu"**, hiraganaChuLvl);  
 hiraganaLvlTable.put(**"hiraganaCho"**, hiraganaChoLvl);  
 hiraganaLvlTable.put(**"hiraganaHya"**, hiraganaHyaLvl);  
 hiraganaLvlTable.put(**"hiraganaHyu"**, hiraganaHyuLvl);  
 hiraganaLvlTable.put(**"hiraganaHyo"**, hiraganaHyoLvl);  
 hiraganaLvlTable.put(**"hiraganaNya"**, hiraganaNyaLvl);  
 hiraganaLvlTable.put(**"hiraganaNyu"**, hiraganaNyuLvl);  
 hiraganaLvlTable.put(**"hiraganaNyo"**, hiraganaNyoLvl);  
 hiraganaLvlTable.put(**"hiraganaMya"**, hiraganaMyaLvl);  
 hiraganaLvlTable.put(**"hiraganaMyu"**, hiraganaMyuLvl);  
 hiraganaLvlTable.put(**"hiraganaMyo"**, hiraganaMyoLvl);  
 hiraganaLvlTable.put(**"hiraganaRya"**, hiraganaRyaLvl);  
 hiraganaLvlTable.put(**"hiraganaRyu"**, hiraganaRyuLvl);  
 hiraganaLvlTable.put(**"hiraganaRyo"**, hiraganaRyoLvl);  
 hiraganaLvlTable.put(**"hiraganaGya"**, hiraganaGyaLvl);  
 hiraganaLvlTable.put(**"hiraganaGyu"**, hiraganaGyuLvl);  
 hiraganaLvlTable.put(**"hiraganaGyo"**, hiraganaGyoLvl);  
 hiraganaLvlTable.put(**"hiraganaJya"**, hiraganaJyaLvl);  
 hiraganaLvlTable.put(**"hiraganaJyu"**, hiraganaJyuLvl);  
 hiraganaLvlTable.put(**"hiraganaJyo"**, hiraganaJyoLvl);  
 hiraganaLvlTable.put(**"hiraganaDya"**, hiraganaDyaLvl);  
 hiraganaLvlTable.put(**"hiraganaDyu"**, hiraganaDyuLvl);  
 hiraganaLvlTable.put(**"hiraganaDyo"**, hiraganaDyoLvl);  
 hiraganaLvlTable.put(**"hiraganaBya"**, hiraganaByaLvl);  
 hiraganaLvlTable.put(**"hiraganaByu"**, hiraganaByuLvl);  
 hiraganaLvlTable.put(**"hiraganaByo"**, hiraganaByoLvl);  
 hiraganaLvlTable.put(**"hiraganaPya"**, hiraganaPyaLvl);  
 hiraganaLvlTable.put(**"hiraganaPyu"**, hiraganaPyuLvl);  
 hiraganaLvlTable.put(**"hiraganaPyo"**, hiraganaPyoLvl);  
  
 **return** hiraganaLvlTable;  
 }  
  
 **public static** Hashtable<String, Integer> getKatakanaLvlTable() {  
  
 katakanaLvlTable = **new** Hashtable<String, Integer>();  
  
 katakanaLvlTable.put(**"katakanaA"**, katakanaALvl);  
 katakanaLvlTable.put(**"katakanaI"**, katakanaILvl);  
 katakanaLvlTable.put(**"katakanaU"**, katakanaULvl);  
 katakanaLvlTable.put(**"katakanaE"**, katakanaELvl);  
 katakanaLvlTable.put(**"katakanaO"**, katakanaOLvl);  
 katakanaLvlTable.put(**"katakanaKa"**, katakanaKaLvl);  
 katakanaLvlTable.put(**"katakanaKi"**, katakanaKiLvl);  
 katakanaLvlTable.put(**"katakanaKu"**, katakanaKuLvl);  
 katakanaLvlTable.put(**"katakanaKe"**, katakanaKeLvl);  
 katakanaLvlTable.put(**"katakanaKo"**, katakanaKoLvl);  
 katakanaLvlTable.put(**"katakanaSa"**, katakanaSaLvl);  
 katakanaLvlTable.put(**"katakanaShi"**, katakanaShiLvl);  
 katakanaLvlTable.put(**"katakanaSu"**, katakanaSuLvl);  
 katakanaLvlTable.put(**"katakanaSe"**, katakanaSeLvl);  
 katakanaLvlTable.put(**"katakanaSo"**, katakanaSoLvl);  
 katakanaLvlTable.put(**"katakanaTa"**, katakanaTaLvl);  
 katakanaLvlTable.put(**"katakanaChi"**, katakanaChiLvl);  
 katakanaLvlTable.put(**"katakanaTsu"**, katakanaTsuLvl);  
 katakanaLvlTable.put(**"katakanaTe"**, katakanaTeLvl);  
 katakanaLvlTable.put(**"katakanaTo"**, katakanaToLvl);  
 katakanaLvlTable.put(**"katakanaNa"**, katakanaNaLvl);  
 katakanaLvlTable.put(**"katakanaNi"**, katakanaNiLvl);  
 katakanaLvlTable.put(**"katakanaNu"**, katakanaNuLvl);  
 katakanaLvlTable.put(**"katakanaNe"**, katakanaNeLvl);  
 katakanaLvlTable.put(**"katakanaNo"**, katakanaNoLvl);  
 katakanaLvlTable.put(**"katakanaHa"**, katakanaHaLvl);  
 katakanaLvlTable.put(**"katakanaHi"**, katakanaHiLvl);  
 katakanaLvlTable.put(**"katakanaHu"**, katakanaHuLvl);  
 katakanaLvlTable.put(**"katakanaHe"**, katakanaHeLvl);  
 katakanaLvlTable.put(**"katakanaHo"**, katakanaHoLvl);  
 katakanaLvlTable.put(**"katakanaMa"**, katakanaMaLvl);  
 katakanaLvlTable.put(**"katakanaMi"**, katakanaMiLvl);  
 katakanaLvlTable.put(**"katakanaMu"**, katakanaMuLvl);  
 katakanaLvlTable.put(**"katakanaMe"**, katakanaMeLvl);  
 katakanaLvlTable.put(**"katakanaMo"**, katakanaMoLvl);  
 katakanaLvlTable.put(**"katakanaRa"**, katakanaRaLvl);  
 katakanaLvlTable.put(**"katakanaRi"**, katakanaRiLvl);  
 katakanaLvlTable.put(**"katakanaRu"**, katakanaRuLvl);  
 katakanaLvlTable.put(**"katakanaRe"**, katakanaReLvl);  
 katakanaLvlTable.put(**"katakanaRo"**, katakanaRoLvl);  
 katakanaLvlTable.put(**"katakanaYu"**, katakanaYuLvl);  
 katakanaLvlTable.put(**"katakanaYo"**, katakanaYoLvl);  
 katakanaLvlTable.put(**"katakanaWa"**, katakanaWaLvl);  
 katakanaLvlTable.put(**"katakanaWo"**, katakanaWoLvl);  
  
 katakanaLvlTable.put(**"katakanaN"**, katakanaNLvl);  
  
 katakanaLvlTable.put(**"katakanaGa"**, katakanaGaLvl);  
 katakanaLvlTable.put(**"katakanaGi"**, katakanaGiLvl);  
 katakanaLvlTable.put(**"katakanaGu"**, katakanaGuLvl);  
 katakanaLvlTable.put(**"katakanaGe"**, katakanaGeLvl);  
 katakanaLvlTable.put(**"katakanaGo"**, katakanaGoLvl);  
 katakanaLvlTable.put(**"katakanaZa"**, katakanaZaLvl);  
 katakanaLvlTable.put(**"katakanaJi"**, katakanaJiLvl);  
 katakanaLvlTable.put(**"katakanaZu"**, katakanaZuLvl);  
 katakanaLvlTable.put(**"katakanaZe"**, katakanaZeLvl);  
 katakanaLvlTable.put(**"katakanaZo"**, katakanaZoLvl);  
 katakanaLvlTable.put(**"katakanaDa"**, katakanaDaLvl);  
 katakanaLvlTable.put(**"katakanaDi"**, katakanaDiLvl);  
 katakanaLvlTable.put(**"katakanaDu"**, katakanaDuLvl);  
 katakanaLvlTable.put(**"katakanaDe"**, katakanaDeLvl);  
 katakanaLvlTable.put(**"katakanaDo"**, katakanaDoLvl);  
 katakanaLvlTable.put(**"katakanaBa"**, katakanaBaLvl);  
 katakanaLvlTable.put(**"katakanaBi"**, katakanaBiLvl);  
 katakanaLvlTable.put(**"katakanaBu"**, katakanaBuLvl);  
 katakanaLvlTable.put(**"katakanaBe"**, katakanaBeLvl);  
 katakanaLvlTable.put(**"katakanaBo"**, katakanaBoLvl);  
 katakanaLvlTable.put(**"katakanaPa"**, katakanaPaLvl);  
 katakanaLvlTable.put(**"katakanaPi"**, katakanaPiLvl);  
 katakanaLvlTable.put(**"katakanaPu"**, katakanaPuLvl);  
 katakanaLvlTable.put(**"katakanaPe"**, katakanaPeLvl);  
 katakanaLvlTable.put(**"katakanaPo"**, katakanaPoLvl);  
 katakanaLvlTable.put(**"katakanaYa"**, katakanaYaLvl);  
 katakanaLvlTable.put(**"katakanaKya"**, katakanaKyaLvl);  
 katakanaLvlTable.put(**"katakanaKyu"**, katakanaKyuLvl);  
 katakanaLvlTable.put(**"katakanaKyo"**, katakanaKyoLvl);  
 katakanaLvlTable.put(**"katakanaSha"**, katakanaShaLvl);  
 katakanaLvlTable.put(**"katakanaShu"**, katakanaShuLvl);  
 katakanaLvlTable.put(**"katakanaSho"**, katakanaShoLvl);  
 katakanaLvlTable.put(**"katakanaCha"**, katakanaChaLvl);  
 katakanaLvlTable.put(**"katakanaChu"**, katakanaChuLvl);  
 katakanaLvlTable.put(**"katakanaCho"**, katakanaChoLvl);  
 katakanaLvlTable.put(**"katakanaHya"**, katakanaHyaLvl);  
 katakanaLvlTable.put(**"katakanaHyu"**, katakanaHyuLvl);  
 katakanaLvlTable.put(**"katakanaHyo"**, katakanaHyoLvl);  
 katakanaLvlTable.put(**"katakanaNya"**, katakanaNyaLvl);  
 katakanaLvlTable.put(**"katakanaNyu"**, katakanaNyuLvl);  
 katakanaLvlTable.put(**"katakanaNyo"**, katakanaNyoLvl);  
 katakanaLvlTable.put(**"katakanaMya"**, katakanaMyaLvl);  
 katakanaLvlTable.put(**"katakanaMyu"**, katakanaMyuLvl);  
 katakanaLvlTable.put(**"katakanaMyo"**, katakanaMyoLvl);  
 katakanaLvlTable.put(**"katakanaRya"**, katakanaRyaLvl);  
 katakanaLvlTable.put(**"katakanaRyu"**, katakanaRyuLvl);  
 katakanaLvlTable.put(**"katakanaRyo"**, katakanaRyoLvl);  
 katakanaLvlTable.put(**"katakanaGya"**, katakanaGyaLvl);  
 katakanaLvlTable.put(**"katakanaGyu"**, katakanaGyuLvl);  
 katakanaLvlTable.put(**"katakanaGyo"**, katakanaGyoLvl);  
 katakanaLvlTable.put(**"katakanaJya"**, katakanaJyaLvl);  
 katakanaLvlTable.put(**"katakanaJyu"**, katakanaJyuLvl);  
 katakanaLvlTable.put(**"katakanaJyo"**, katakanaJyoLvl);  
 katakanaLvlTable.put(**"katakanaDya"**, katakanaDyaLvl);  
 katakanaLvlTable.put(**"katakanaDyu"**, katakanaDyuLvl);  
 katakanaLvlTable.put(**"katakanaDyo"**, katakanaDyoLvl);  
 katakanaLvlTable.put(**"katakanaBya"**, katakanaByaLvl);  
 katakanaLvlTable.put(**"katakanaByu"**, katakanaByuLvl);  
 katakanaLvlTable.put(**"katakanaByo"**, katakanaByoLvl);  
 katakanaLvlTable.put(**"katakanaPya"**, katakanaPyaLvl);  
 katakanaLvlTable.put(**"katakanaPyu"**, katakanaPyuLvl);  
 katakanaLvlTable.put(**"katakanaPyo"**, katakanaPyoLvl);  
  
 **return** katakanaLvlTable;  
 }  
  
 **public static** Hashtable<String, Integer> getKanjiLvlTable() {  
  
 kanjiLvlTable = **new** Hashtable<String, Integer>();  
  
 kanjiLvlTable.put(**"kanjiHa"**, kanjiHaLvl);  
 kanjiLvlTable.put(**"kanjiKi"**, kanjiKiLvl);  
 kanjiLvlTable.put(**"kanjiMushi"**, kanjiMushiLvl);  
 kanjiLvlTable.put(**"kanjiKaze"**, kanjiKazeLvl);  
 kanjiLvlTable.put(**"kanjiTsuchi"**, kanjiTsuchiLvl);  
 kanjiLvlTable.put(**"kanjiUchi"**, kanjiUchiLvl);  
 kanjiLvlTable.put(**"kanjiUmi"**, kanjiUmiLvl);  
 kanjiLvlTable.put(**"kanjiMizu"**, kanjiMizuLvl);  
 kanjiLvlTable.put(**"kanjiIke"**, kanjiIkeLvl);  
 kanjiLvlTable.put(**"kanjiKoori"**, kanjiKooriLvl);  
 kanjiLvlTable.put(**"kanjiHayashi"**, kanjiHayashiLvl);  
 kanjiLvlTable.put(**"kanjiTo"**, kanjiToLvl);  
 kanjiLvlTable.put(**"kanjiTera"**, kanjiTeraLvl);  
 kanjiLvlTable.put(**"kanjiSoto"**, kanjiSotoLvl);  
 kanjiLvlTable.put(**"kanjiHi"**, kanjiHiLvl);  
 kanjiLvlTable.put(**"kanjiAka"**, kanjiAkaLvl);  
 kanjiLvlTable.put(**"kanjiHana"**, kanjiHanaLvl);  
 kanjiLvlTable.put(**"kanjiMori"**, kanjiMoriLvl);  
 kanjiLvlTable.put(**"kanjiIshi"**, kanjiIshiLvl);  
 kanjiLvlTable.put(**"kanjiYama"**, kanjiYamaLvl);  
 kanjiLvlTable.put(**"kanjiSakana"**, kanjiSakanaLvl);  
 kanjiLvlTable.put(**"kanjiKai"**, kanjiKaiLvl);  
 kanjiLvlTable.put(**"kanjiYuki"**, kanjiYukiLvl);  
 kanjiLvlTable.put(**"kanjiKawa"**, kanjiKawaLvl);  
 kanjiLvlTable.put(**"kanjiFuyu"**, kanjiFuyuLvl);  
 kanjiLvlTable.put(**"kanjiMon"**, kanjiMonLvl);  
 kanjiLvlTable.put(**"kanjiAna"**, kanjiAnaLvl);  
 kanjiLvlTable.put(**"kanjiKa"**, kanjiKaLvl);  
 kanjiLvlTable.put(**"kanjiChi"**, kanjiChiLvl);  
 kanjiLvlTable.put(**"kanjiHikari"**, kanjiHikariLvl);  
  
 **return** kanjiLvlTable;  
 }  
  
 **public static void** increaseLvl(String letter, **int** addOn){  
 **switch** (letter){  
 **case "hiraganaA"**: hiraganaALvl = MathUtils.clamp(addOn + hiraganaALvl, 1, 4); **break**;  
 **case "hiraganaI"**: hiraganaILvl = MathUtils.clamp(addOn + hiraganaILvl, 1, 4); **break**;  
 **case "hiraganaU"**: hiraganaULvl = MathUtils.clamp(addOn + hiraganaULvl, 1, 4); **break**;  
 **case "hiraganaE"**: hiraganaELvl = MathUtils.clamp(addOn + hiraganaELvl, 1, 4); **break**;  
 **case "hiraganaO"**: hiraganaOLvl = MathUtils.clamp(addOn + hiraganaOLvl, 1, 4); **break**;  
 **case "hiraganaKa"**: hiraganaKaLvl = MathUtils.clamp(addOn + hiraganaKaLvl, 1, 4); **break**;  
 **case "hiraganaKi"**: hiraganaKiLvl = MathUtils.clamp(addOn + hiraganaKiLvl, 1, 4); **break**;  
 **case "hiraganaKu"**: hiraganaKuLvl = MathUtils.clamp(addOn + hiraganaKuLvl, 1, 4); **break**;  
 **case "hiraganaKe"**: hiraganaKeLvl = MathUtils.clamp(addOn + hiraganaKeLvl, 1, 4); **break**;  
 **case "hiraganaKo"**: hiraganaKoLvl = MathUtils.clamp(addOn + hiraganaKoLvl, 1, 4); **break**;  
 **case "hiraganaSa"**: hiraganaSaLvl = MathUtils.clamp(addOn + hiraganaSaLvl, 1, 4); **break**;  
 **case "hiraganaShi"**: hiraganaShiLvl = MathUtils.clamp(addOn + hiraganaShiLvl, 1, 4); **break**;  
 **case "hiraganaSu"**: hiraganaSuLvl = MathUtils.clamp(addOn + hiraganaSuLvl, 1, 4); **break**;  
 **case "hiraganaSe"**: hiraganaSeLvl = MathUtils.clamp(addOn + hiraganaSeLvl, 1, 4); **break**;  
 **case "hiraganaSo"**: hiraganaSoLvl = MathUtils.clamp(addOn + hiraganaSoLvl, 1, 4); **break**;  
 **case "hiraganaTa"**: hiraganaTaLvl = MathUtils.clamp(addOn + hiraganaTaLvl, 1, 4); **break**;  
 **case "hiraganaChi"**: hiraganaChiLvl = MathUtils.clamp(addOn + hiraganaChiLvl, 1, 4); **break**;  
 **case "hiraganaTsu"**: hiraganaTsuLvl = MathUtils.clamp(addOn + hiraganaTsuLvl, 1, 4); **break**;  
 **case "hiraganaTe"**: hiraganaTeLvl = MathUtils.clamp(addOn + hiraganaTeLvl, 1, 4); **break**;  
 **case "hiraganaTo"**: hiraganaToLvl = MathUtils.clamp(addOn + hiraganaToLvl, 1, 4); **break**;  
 **case "hiraganaNa"**: hiraganaNaLvl = MathUtils.clamp(addOn + hiraganaNaLvl, 1, 4); **break**;  
 **case "hiraganaNi"**: hiraganaNiLvl = MathUtils.clamp(addOn + hiraganaNiLvl, 1, 4); **break**;  
 **case "hiraganaNu"**: hiraganaNuLvl = MathUtils.clamp(addOn + hiraganaNuLvl, 1, 4); **break**;  
 **case "hiraganaNe"**: hiraganaNeLvl = MathUtils.clamp(addOn + hiraganaNeLvl, 1, 4); **break**;  
 **case "hiraganaNo"**: hiraganaNoLvl = MathUtils.clamp(addOn + hiraganaNoLvl, 1, 4); **break**;  
 **case "hiraganaHa"**: hiraganaHaLvl = MathUtils.clamp(addOn + hiraganaHaLvl, 1, 4); **break**;  
 **case "hiraganaHi"**: hiraganaHiLvl = MathUtils.clamp(addOn + hiraganaHiLvl, 1, 4); **break**;  
 **case "hiraganaHu"**: hiraganaHuLvl = MathUtils.clamp(addOn + hiraganaHuLvl, 1, 4); **break**;  
 **case "hiraganaHe"**: hiraganaHeLvl = MathUtils.clamp(addOn + hiraganaHeLvl, 1, 4); **break**;  
 **case "hiraganaHo"**: hiraganaHoLvl = MathUtils.clamp(addOn + hiraganaHoLvl, 1, 4); **break**;  
 **case "hiraganaMa"**: hiraganaMaLvl = MathUtils.clamp(addOn + hiraganaMaLvl, 1, 4); **break**;  
 **case "hiraganaMi"**: hiraganaMiLvl = MathUtils.clamp(addOn + hiraganaMiLvl, 1, 4); **break**;  
 **case "hiraganaMu"**: hiraganaMuLvl = MathUtils.clamp(addOn + hiraganaMuLvl, 1, 4); **break**;  
 **case "hiraganaMe"**: hiraganaMeLvl = MathUtils.clamp(addOn + hiraganaMeLvl, 1, 4); **break**;  
 **case "hiraganaMo"**: hiraganaMoLvl = MathUtils.clamp(addOn + hiraganaMoLvl, 1, 4); **break**;  
 **case "hiraganaRa"**: hiraganaRaLvl = MathUtils.clamp(addOn + hiraganaRaLvl, 1, 4); **break**;  
 **case "hiraganaRi"**: hiraganaRiLvl = MathUtils.clamp(addOn + hiraganaRiLvl, 1, 4); **break**;  
 **case "hiraganaRu"**: hiraganaRuLvl = MathUtils.clamp(addOn + hiraganaRuLvl, 1, 4); **break**;  
 **case "hiraganaRe"**: hiraganaReLvl = MathUtils.clamp(addOn + hiraganaReLvl, 1, 4); **break**;  
 **case "hiraganaRo"**: hiraganaRoLvl = MathUtils.clamp(addOn + hiraganaRoLvl, 1, 4); **break**;  
 **case "hiraganaYa"**: hiraganaYaLvl = MathUtils.clamp(addOn + hiraganaYaLvl, 1, 4); **break**;  
 **case "hiraganaYu"**: hiraganaYuLvl = MathUtils.clamp(addOn + hiraganaYuLvl, 1, 4); **break**;  
 **case "hiraganaYo"**: hiraganaYoLvl = MathUtils.clamp(addOn + hiraganaYoLvl, 1, 4); **break**;  
 **case "hiraganaWa"**: hiraganaWaLvl = MathUtils.clamp(addOn + hiraganaWaLvl, 1, 4); **break**;  
 **case "hiraganaWo"**: hiraganaWoLvl = MathUtils.clamp(addOn + hiraganaWoLvl, 1, 4); **break**;  
 **case "hiraganaN"**: hiraganaNLvl = MathUtils.clamp(addOn + hiraganaNLvl, 1, 4); **break**;  
 **case "hiraganaGa"**: hiraganaGaLvl = MathUtils.clamp(addOn + hiraganaGaLvl, 1, 4); **break**;  
 **case "hiraganaGi"**: hiraganaGiLvl = MathUtils.clamp(addOn + hiraganaGiLvl, 1, 4); **break**;  
 **case "hiraganaGu"**: hiraganaGuLvl = MathUtils.clamp(addOn + hiraganaGuLvl, 1, 4); **break**;  
 **case "hiraganaGe"**: hiraganaGeLvl = MathUtils.clamp(addOn + hiraganaGeLvl, 1, 4); **break**;  
 **case "hiraganaGo"**: hiraganaGoLvl = MathUtils.clamp(addOn + hiraganaGoLvl, 1, 4); **break**;  
 **case "hiraganaZa"**: hiraganaZaLvl = MathUtils.clamp(addOn + hiraganaZaLvl, 1, 4); **break**;  
 **case "hiraganaJi"**: hiraganaJiLvl = MathUtils.clamp(addOn + hiraganaJiLvl, 1, 4); **break**;  
 **case "hiraganaZu"**: hiraganaZuLvl = MathUtils.clamp(addOn + hiraganaZuLvl, 1, 4); **break**;  
 **case "hiraganaZe"**: hiraganaZeLvl = MathUtils.clamp(addOn + hiraganaZeLvl, 1, 4); **break**;  
 **case "hiraganaZo"**: hiraganaZoLvl = MathUtils.clamp(addOn + hiraganaZoLvl, 1, 4); **break**;  
 **case "hiraganaDa"**: hiraganaDaLvl = MathUtils.clamp(addOn + hiraganaDaLvl, 1, 4); **break**;  
 **case "hiraganaDi"**: hiraganaDiLvl = MathUtils.clamp(addOn + hiraganaDiLvl, 1, 4); **break**;  
 **case "hiraganaDu"**: hiraganaDuLvl = MathUtils.clamp(addOn + hiraganaDuLvl, 1, 4); **break**;  
 **case "hiraganaDe"**: hiraganaDeLvl = MathUtils.clamp(addOn + hiraganaDeLvl, 1, 4); **break**;  
 **case "hiraganaDo"**: hiraganaDoLvl = MathUtils.clamp(addOn + hiraganaDoLvl, 1, 4); **break**;  
 **case "hiraganaBa"**: hiraganaBaLvl = MathUtils.clamp(addOn + hiraganaBaLvl, 1, 4); **break**;  
 **case "hiraganaBi"**: hiraganaBiLvl = MathUtils.clamp(addOn + hiraganaBiLvl, 1, 4); **break**;  
 **case "hiraganaBu"**: hiraganaBuLvl = MathUtils.clamp(addOn + hiraganaBuLvl, 1, 4); **break**;  
 **case "hiraganaBe"**: hiraganaBeLvl = MathUtils.clamp(addOn + hiraganaBeLvl, 1, 4); **break**;  
 **case "hiraganaBo"**: hiraganaBoLvl = MathUtils.clamp(addOn + hiraganaBoLvl, 1, 4); **break**;  
 **case "hiraganaPa"**: hiraganaPaLvl = MathUtils.clamp(addOn + hiraganaPaLvl, 1, 4); **break**;  
 **case "hiraganaPi"**: hiraganaPiLvl = MathUtils.clamp(addOn + hiraganaPiLvl, 1, 4); **break**;  
 **case "hiraganaPu"**: hiraganaPuLvl = MathUtils.clamp(addOn + hiraganaPuLvl, 1, 4); **break**;  
 **case "hiraganaPe"**: hiraganaPeLvl = MathUtils.clamp(addOn + hiraganaPeLvl, 1, 4); **break**;  
 **case "hiraganaPo"**: hiraganaPoLvl = MathUtils.clamp(addOn + hiraganaPoLvl, 1, 4); **break**;  
 **case "hiraganaKya"**: hiraganaKyaLvl = MathUtils.clamp(addOn + hiraganaKyaLvl, 1, 4); **break**;  
 **case "hiraganaKyu"**: hiraganaKyuLvl = MathUtils.clamp(addOn + hiraganaKyuLvl, 1, 4); **break**;  
 **case "hiraganaKyo"**: hiraganaKyoLvl = MathUtils.clamp(addOn + hiraganaKyoLvl, 1, 4); **break**;  
 **case "hiraganaSha"**: hiraganaShaLvl = MathUtils.clamp(addOn + hiraganaShaLvl, 1, 4); **break**;  
 **case "hiraganaShu"**: hiraganaShuLvl = MathUtils.clamp(addOn + hiraganaShuLvl, 1, 4); **break**;  
 **case "hiraganaSho"**: hiraganaShoLvl = MathUtils.clamp(addOn + hiraganaShoLvl, 1, 4); **break**;  
 **case "hiraganaCha"**: hiraganaChaLvl = MathUtils.clamp(addOn + hiraganaChaLvl, 1, 4); **break**;  
 **case "hiraganaChu"**: hiraganaChuLvl = MathUtils.clamp(addOn + hiraganaChuLvl, 1, 4); **break**;  
 **case "hiraganaCho"**: hiraganaChoLvl = MathUtils.clamp(addOn + hiraganaChoLvl, 1, 4); **break**;  
 **case "hiraganaHya"**: hiraganaHyaLvl = MathUtils.clamp(addOn + hiraganaHyaLvl, 1, 4); **break**;  
 **case "hiraganaHyu"**: hiraganaHyuLvl = MathUtils.clamp(addOn + hiraganaHyuLvl, 1, 4); **break**;  
 **case "hiraganaHyo"**: hiraganaHyoLvl = MathUtils.clamp(addOn + hiraganaHyoLvl, 1, 4); **break**;  
 **case "hiraganaNya"**: hiraganaNyaLvl = MathUtils.clamp(addOn + hiraganaNyaLvl, 1, 4); **break**;  
 **case "hiraganaNyu"**: hiraganaNyuLvl = MathUtils.clamp(addOn + hiraganaNyuLvl, 1, 4); **break**;  
 **case "hiraganaNyo"**: hiraganaNyoLvl = MathUtils.clamp(addOn + hiraganaNyoLvl, 1, 4); **break**;  
 **case "hiraganaMya"**: hiraganaMyaLvl = MathUtils.clamp(addOn + hiraganaMyaLvl, 1, 4); **break**;  
 **case "hiraganaMyu"**: hiraganaMyuLvl = MathUtils.clamp(addOn + hiraganaMyuLvl, 1, 4); **break**;  
 **case "hiraganaMyo"**: hiraganaMyoLvl = MathUtils.clamp(addOn + hiraganaMyoLvl, 1, 4); **break**;  
 **case "hiraganaRya"**: hiraganaRyaLvl = MathUtils.clamp(addOn + hiraganaRyaLvl, 1, 4); **break**;  
 **case "hiraganaRyu"**: hiraganaRyuLvl = MathUtils.clamp(addOn + hiraganaRyuLvl, 1, 4); **break**;  
 **case "hiraganaRyo"**: hiraganaRyoLvl = MathUtils.clamp(addOn + hiraganaRyoLvl, 1, 4); **break**;  
 **case "hiraganaGya"**: hiraganaGyaLvl = MathUtils.clamp(addOn + hiraganaGyaLvl, 1, 4); **break**;  
 **case "hiraganaGyu"**: hiraganaGyuLvl = MathUtils.clamp(addOn + hiraganaGyuLvl, 1, 4); **break**;  
 **case "hiraganaGyo"**: hiraganaGyoLvl = MathUtils.clamp(addOn + hiraganaGyoLvl, 1, 4); **break**;  
 **case "hiraganaJya"**: hiraganaJyaLvl = MathUtils.clamp(addOn + hiraganaJyaLvl, 1, 4); **break**;  
 **case "hiraganaJyu"**: hiraganaJyuLvl = MathUtils.clamp(addOn + hiraganaJyuLvl, 1, 4); **break**;  
 **case "hiraganaJyo"**: hiraganaJyoLvl = MathUtils.clamp(addOn + hiraganaJyoLvl, 1, 4); **break**;  
 **case "hiraganaDya"**: hiraganaDyaLvl = MathUtils.clamp(addOn + hiraganaDyaLvl, 1, 4); **break**;  
 **case "hiraganaDyu"**: hiraganaDyuLvl = MathUtils.clamp(addOn + hiraganaDyuLvl, 1, 4); **break**;  
 **case "hiraganaDyo"**: hiraganaDyoLvl = MathUtils.clamp(addOn + hiraganaDyoLvl, 1, 4); **break**;  
 **case "hiraganaBya"**: hiraganaByaLvl = MathUtils.clamp(addOn + hiraganaByaLvl, 1, 4); **break**;  
 **case "hiraganaByu"**: hiraganaByuLvl = MathUtils.clamp(addOn + hiraganaByuLvl, 1, 4); **break**;  
 **case "hiraganaByo"**: hiraganaByoLvl = MathUtils.clamp(addOn + hiraganaByoLvl, 1, 4); **break**;  
 **case "hiraganaPya"**: hiraganaPyaLvl = MathUtils.clamp(addOn + hiraganaPyaLvl, 1, 4); **break**;  
 **case "hiraganaPyu"**: hiraganaPyuLvl = MathUtils.clamp(addOn + hiraganaPyuLvl, 1, 4); **break**;  
 **case "hiraganaPyo"**: hiraganaPyoLvl = MathUtils.clamp(addOn + hiraganaPyoLvl, 1, 4); **break**;  
 **case "katakanaA"**: katakanaALvl = MathUtils.clamp(addOn + katakanaALvl, 1, 4); **break**;  
 **case "katakanaI"**: katakanaILvl = MathUtils.clamp(addOn + katakanaILvl, 1, 4); **break**;  
 **case "katakanaU"**: katakanaULvl = MathUtils.clamp(addOn + katakanaULvl, 1, 4); **break**;  
 **case "katakanaE"**: katakanaELvl = MathUtils.clamp(addOn + katakanaELvl, 1, 4); **break**;  
 **case "katakanaO"**: katakanaOLvl = MathUtils.clamp(addOn + katakanaOLvl, 1, 4); **break**;  
 **case "katakanaKa"**: katakanaKaLvl = MathUtils.clamp(addOn + katakanaKaLvl, 1, 4); **break**;  
 **case "katakanaKi"**: katakanaKiLvl = MathUtils.clamp(addOn + katakanaKiLvl, 1, 4); **break**;  
 **case "katakanaKu"**: katakanaKuLvl = MathUtils.clamp(addOn + katakanaKuLvl, 1, 4); **break**;  
 **case "katakanaKe"**: katakanaKeLvl = MathUtils.clamp(addOn + katakanaKeLvl, 1, 4); **break**;  
 **case "katakanaKo"**: katakanaKoLvl = MathUtils.clamp(addOn + katakanaKoLvl, 1, 4); **break**;  
 **case "katakanaSa"**: katakanaSaLvl = MathUtils.clamp(addOn + katakanaSaLvl, 1, 4); **break**;  
 **case "katakanaLvlShi"**: katakanaShiLvl = MathUtils.clamp(addOn + katakanaShiLvl, 1, 4); **break**;  
 **case "katakanaLvlSu"**: katakanaSuLvl = MathUtils.clamp(addOn + katakanaSuLvl, 1, 4); **break**;  
 **case "katakanaLvlSe"**: katakanaSeLvl = MathUtils.clamp(addOn + katakanaSeLvl, 1, 4); **break**;  
 **case "katakanaLvlSo"**: katakanaSoLvl = MathUtils.clamp(addOn + katakanaSoLvl, 1, 4); **break**;  
 **case "katakanaLvlTa"**: katakanaTaLvl = MathUtils.clamp(addOn + katakanaTaLvl, 1, 4); **break**;  
 **case "katakanaLvlChi"**: katakanaChiLvl = MathUtils.clamp(addOn + katakanaChiLvl, 1, 4); **break**;  
 **case "katakanaLvlTsu"**: katakanaTsuLvl = MathUtils.clamp(addOn + katakanaTsuLvl, 1, 4); **break**;  
 **case "katakanaLvlTe"**: katakanaTeLvl = MathUtils.clamp(addOn + katakanaTeLvl, 1, 4); **break**;  
 **case "katakanaLvlTo"**: katakanaToLvl = MathUtils.clamp(addOn + katakanaToLvl, 1, 4); **break**;  
 **case "katakanaLvlNa"**: katakanaNaLvl = MathUtils.clamp(addOn + katakanaNaLvl, 1, 4); **break**;  
 **case "katakanaLvlNi"**: katakanaNiLvl = MathUtils.clamp(addOn + katakanaNiLvl, 1, 4); **break**;  
 **case "katakanaLvlNu"**: katakanaNuLvl = MathUtils.clamp(addOn + katakanaNuLvl, 1, 4); **break**;  
 **case "katakanaLvlNe"**: katakanaNeLvl = MathUtils.clamp(addOn + katakanaNeLvl, 1, 4); **break**;  
 **case "katakanaLvlNo"**: katakanaNoLvl = MathUtils.clamp(addOn + katakanaNoLvl, 1, 4); **break**;  
 **case "katakanaLvlHa"**: katakanaHaLvl = MathUtils.clamp(addOn + katakanaHaLvl, 1, 4); **break**;  
 **case "katakanaLvlHi"**: katakanaHiLvl = MathUtils.clamp(addOn + katakanaHiLvl, 1, 4); **break**;  
 **case "katakanaLvlHu"**: katakanaHuLvl = MathUtils.clamp(addOn + katakanaHuLvl, 1, 4); **break**;  
 **case "katakanaLvlHe"**: katakanaHeLvl = MathUtils.clamp(addOn + katakanaHeLvl, 1, 4); **break**;  
 **case "katakanaLvlHo"**: katakanaHoLvl = MathUtils.clamp(addOn + katakanaHoLvl, 1, 4); **break**;  
 **case "katakanaLvlMa"**: katakanaMaLvl = MathUtils.clamp(addOn + katakanaMaLvl, 1, 4); **break**;  
 **case "katakanaLvlMi"**: katakanaMiLvl = MathUtils.clamp(addOn + katakanaMiLvl, 1, 4); **break**;  
 **case "katakanaLvlMu"**: katakanaMuLvl = MathUtils.clamp(addOn + katakanaMuLvl, 1, 4); **break**;  
 **case "katakanaLvlMe"**: katakanaMeLvl = MathUtils.clamp(addOn + katakanaMeLvl, 1, 4); **break**;  
 **case "katakanaLvlMo"**: katakanaMoLvl = MathUtils.clamp(addOn + katakanaMoLvl, 1, 4); **break**;  
 **case "katakanaLvlRa"**: katakanaRaLvl = MathUtils.clamp(addOn + katakanaRaLvl, 1, 4); **break**;  
 **case "katakanaLvlRi"**: katakanaRiLvl = MathUtils.clamp(addOn + katakanaRiLvl, 1, 4); **break**;  
 **case "katakanaLvlRu"**: katakanaRuLvl = MathUtils.clamp(addOn + katakanaRuLvl, 1, 4); **break**;  
 **case "katakanaLvlRe"**: katakanaReLvl = MathUtils.clamp(addOn + katakanaReLvl, 1, 4); **break**;  
 **case "katakanaLvlRo"**: katakanaRoLvl = MathUtils.clamp(addOn + katakanaRoLvl, 1, 4); **break**;  
 **case "katakanaLvlYa"**: katakanaYaLvl = MathUtils.clamp(addOn + katakanaYaLvl, 1, 4); **break**;  
 **case "katakanaLvlYu"**: katakanaYuLvl = MathUtils.clamp(addOn + katakanaYuLvl, 1, 4); **break**;  
 **case "katakanaLvlYo"**: katakanaYoLvl = MathUtils.clamp(addOn + katakanaYoLvl, 1, 4); **break**;  
 **case "katakanaLvlWa"**: katakanaWaLvl = MathUtils.clamp(addOn + katakanaWaLvl, 1, 4); **break**;  
 **case "katakanaLvlWo"**: katakanaWoLvl = MathUtils.clamp(addOn + katakanaWoLvl, 1, 4); **break**;  
 **case "katakanaLvlN"**: katakanaNLvl = MathUtils.clamp(addOn + katakanaNLvl, 1, 4); **break**;  
 **case "katakanaLvlGa"**: katakanaGaLvl = MathUtils.clamp(addOn + katakanaGaLvl, 1, 4); **break**;  
 **case "katakanaLvlGi"**: katakanaGiLvl = MathUtils.clamp(addOn + katakanaGiLvl, 1, 4); **break**;  
 **case "katakanaLvlGu"**: katakanaGuLvl = MathUtils.clamp(addOn + katakanaGuLvl, 1, 4); **break**;  
 **case "katakanaLvlGe"**: katakanaGeLvl = MathUtils.clamp(addOn + katakanaGeLvl, 1, 4); **break**;  
 **case "katakanaLvlGo"**: katakanaGoLvl = MathUtils.clamp(addOn + katakanaGoLvl, 1, 4); **break**;  
 **case "katakanaLvlZa"**: katakanaZaLvl = MathUtils.clamp(addOn + katakanaZaLvl, 1, 4); **break**;  
 **case "katakanaLvlJi"**: katakanaJiLvl = MathUtils.clamp(addOn + katakanaJiLvl, 1, 4); **break**;  
 **case "katakanaLvlZu"**: katakanaZuLvl = MathUtils.clamp(addOn + katakanaZuLvl, 1, 4); **break**;  
 **case "katakanaLvlZe"**: katakanaZeLvl = MathUtils.clamp(addOn + katakanaZeLvl, 1, 4); **break**;  
 **case "katakanaLvlZo"**: katakanaZoLvl = MathUtils.clamp(addOn + katakanaZoLvl, 1, 4); **break**;  
 **case "katakanaLvlDa"**: katakanaDaLvl = MathUtils.clamp(addOn + katakanaDaLvl, 1, 4); **break**;  
 **case "katakanaLvlDi"**: katakanaDiLvl = MathUtils.clamp(addOn + katakanaDiLvl, 1, 4); **break**;  
 **case "katakanaLvlDu"**: katakanaDuLvl = MathUtils.clamp(addOn + katakanaDuLvl, 1, 4); **break**;  
 **case "katakanaLvlDe"**: katakanaDeLvl = MathUtils.clamp(addOn + katakanaDeLvl, 1, 4); **break**;  
 **case "katakanaLvlDo"**: katakanaDoLvl = MathUtils.clamp(addOn + katakanaDoLvl, 1, 4); **break**;  
 **case "katakanaLvlBa"**: katakanaBaLvl = MathUtils.clamp(addOn + katakanaBaLvl, 1, 4); **break**;  
 **case "katakanaLvlBi"**: katakanaBiLvl = MathUtils.clamp(addOn + katakanaBiLvl, 1, 4); **break**;  
 **case "katakanaLvlBu"**: katakanaBuLvl = MathUtils.clamp(addOn + katakanaBuLvl, 1, 4); **break**;  
 **case "katakanaLvlBe"**: katakanaBeLvl = MathUtils.clamp(addOn + katakanaBeLvl, 1, 4); **break**;  
 **case "katakanaLvlBo"**: katakanaBoLvl = MathUtils.clamp(addOn + katakanaBoLvl, 1, 4); **break**;  
 **case "katakanaLvlPa"**: katakanaPaLvl = MathUtils.clamp(addOn + katakanaPaLvl, 1, 4); **break**;  
 **case "katakanaLvlPi"**: katakanaPiLvl = MathUtils.clamp(addOn + katakanaPiLvl, 1, 4); **break**;  
 **case "katakanaLvlPu"**: katakanaPuLvl = MathUtils.clamp(addOn + katakanaPuLvl, 1, 4); **break**;  
 **case "katakanaLvlPe"**: katakanaPeLvl = MathUtils.clamp(addOn + katakanaPeLvl, 1, 4); **break**;  
 **case "katakanaLvlPo"**: katakanaPoLvl = MathUtils.clamp(addOn + katakanaPoLvl, 1, 4); **break**;  
 **case "katakanaLvlKya"**: katakanaKyaLvl = MathUtils.clamp(addOn + katakanaKyaLvl, 1, 4); **break**;  
 **case "katakanaLvlKyu"**: katakanaKyuLvl = MathUtils.clamp(addOn + katakanaKyuLvl, 1, 4); **break**;  
 **case "katakanaLvlKyo"**: katakanaKyoLvl = MathUtils.clamp(addOn + katakanaKyoLvl, 1, 4); **break**;  
 **case "katakanaLvlSha"**: katakanaShaLvl = MathUtils.clamp(addOn + katakanaShaLvl, 1, 4); **break**;  
 **case "katakanaLvlShu"**: katakanaShuLvl = MathUtils.clamp(addOn + katakanaShuLvl, 1, 4); **break**;  
 **case "katakanaLvlSho"**: katakanaShoLvl = MathUtils.clamp(addOn + katakanaShoLvl, 1, 4); **break**;  
 **case "katakanaLvlCha"**: katakanaChaLvl = MathUtils.clamp(addOn + katakanaChaLvl, 1, 4); **break**;  
 **case "katakanaLvlChu"**: katakanaChuLvl = MathUtils.clamp(addOn + katakanaChuLvl, 1, 4); **break**;  
 **case "katakanaLvlCho"**: katakanaChoLvl = MathUtils.clamp(addOn + katakanaChoLvl, 1, 4); **break**;  
 **case "katakanaLvlHya"**: katakanaHyaLvl = MathUtils.clamp(addOn + katakanaHyaLvl, 1, 4); **break**;  
 **case "katakanaLvlHyu"**: katakanaHyuLvl = MathUtils.clamp(addOn + katakanaHyuLvl, 1, 4); **break**;  
 **case "katakanaLvlHyo"**: katakanaHyoLvl = MathUtils.clamp(addOn + katakanaHyoLvl, 1, 4); **break**;  
 **case "katakanaLvlNya"**: katakanaNyaLvl = MathUtils.clamp(addOn + katakanaNyaLvl, 1, 4); **break**;  
 **case "katakanaLvlNyu"**: katakanaNyuLvl = MathUtils.clamp(addOn + katakanaNyuLvl, 1, 4); **break**;  
 **case "katakanaLvlNyo"**: katakanaNyoLvl = MathUtils.clamp(addOn + katakanaNyoLvl, 1, 4); **break**;  
 **case "katakanaLvlMya"**: katakanaMyaLvl = MathUtils.clamp(addOn + katakanaMyaLvl, 1, 4); **break**;  
 **case "katakanaLvlMyu"**: katakanaMyuLvl = MathUtils.clamp(addOn + katakanaMyuLvl, 1, 4); **break**;  
 **case "katakanaLvlMyo"**: katakanaMyoLvl = MathUtils.clamp(addOn + katakanaMyoLvl, 1, 4); **break**;  
 **case "katakanaLvlRya"**: katakanaRyaLvl = MathUtils.clamp(addOn + katakanaRyaLvl, 1, 4); **break**;  
 **case "katakanaLvlRyu"**: katakanaRyuLvl = MathUtils.clamp(addOn + katakanaRyuLvl, 1, 4); **break**;  
 **case "katakanaLvlRyo"**: katakanaRyoLvl = MathUtils.clamp(addOn + katakanaRyoLvl, 1, 4); **break**;  
 **case "katakanaLvlGya"**: katakanaGyaLvl = MathUtils.clamp(addOn + katakanaGyaLvl, 1, 4); **break**;  
 **case "katakanaLvlGyu"**: katakanaGyuLvl = MathUtils.clamp(addOn + katakanaGyuLvl, 1, 4); **break**;  
 **case "katakanaLvlGyo"**: katakanaGyoLvl = MathUtils.clamp(addOn + katakanaGyoLvl, 1, 4); **break**;  
 **case "katakanaLvlJya"**: katakanaJyaLvl = MathUtils.clamp(addOn + katakanaJyaLvl, 1, 4); **break**;  
 **case "katakanaLvlJyu"**: katakanaJyuLvl = MathUtils.clamp(addOn + katakanaJyuLvl, 1, 4); **break**;  
 **case "katakanaLvlJyo"**: katakanaJyoLvl = MathUtils.clamp(addOn + katakanaJyoLvl, 1, 4); **break**;  
 **case "katakanaLvlDya"**: katakanaDyaLvl = MathUtils.clamp(addOn + katakanaDyaLvl, 1, 4); **break**;  
 **case "katakanaLvlDyu"**: katakanaDyuLvl = MathUtils.clamp(addOn + katakanaDyuLvl, 1, 4); **break**;  
 **case "katakanaLvlDyo"**: katakanaDyoLvl = MathUtils.clamp(addOn + katakanaDyoLvl, 1, 4); **break**;  
 **case "katakanaLvlBya"**: katakanaByaLvl = MathUtils.clamp(addOn + katakanaByaLvl, 1, 4); **break**;  
 **case "katakanaLvlByu"**: katakanaByuLvl = MathUtils.clamp(addOn + katakanaByuLvl, 1, 4); **break**;  
 **case "katakanaLvlByo"**: katakanaByoLvl = MathUtils.clamp(addOn + katakanaByoLvl, 1, 4); **break**;  
 **case "katakanaLvlPya"**: katakanaPyaLvl = MathUtils.clamp(addOn + katakanaPyaLvl, 1, 4); **break**;  
 **case "katakanaLvlPyu"**: katakanaPyuLvl = MathUtils.clamp(addOn + katakanaPyuLvl, 1, 4); **break**;  
 **case "katakanaLvlPyo"**: katakanaPyoLvl = MathUtils.clamp(addOn + katakanaPyoLvl, 1, 4); **break**;  
 **case "kanjiHa"**: kanjiHaLvl = MathUtils.clamp(addOn + kanjiHaLvl, 1, 6); **break**;  
 **case "kanjiKi"**: kanjiKiLvl = MathUtils.clamp(addOn + kanjiKiLvl, 1, 6); **break**;  
 **case "kanjiMushi"**: kanjiMushiLvl = MathUtils.clamp(addOn + kanjiMushiLvl, 1, 6); **break**;  
 **case "kanjiKaze"**: kanjiKazeLvl = MathUtils.clamp(addOn + kanjiKazeLvl, 1, 6); **break**;  
 **case "kanjiUchi"**: kanjiUchiLvl = MathUtils.clamp(addOn + kanjiUchiLvl, 1, 6); **break**;  
 **case "kanjiUmi"**: kanjiUmiLvl = MathUtils.clamp(addOn + kanjiUmiLvl, 1, 6); **break**;  
 **case "kanjiMizu"**: kanjiMizuLvl = MathUtils.clamp(addOn + kanjiMizuLvl, 1, 6); **break**;  
 **case "kanjiIke"**: kanjiIkeLvl = MathUtils.clamp(addOn + kanjiIkeLvl, 1, 6); **break**;  
 **case "kanjiKoori"**: kanjiKooriLvl = MathUtils.clamp(addOn + kanjiKooriLvl, 1, 6); **break**;  
 **case "kanjiHayashi"**:kanjiHayashiLvl = MathUtils.clamp(addOn + kanjiHayashiLvl, 1, 6); **break**;  
 **case "kanjiTo"**: kanjiToLvl = MathUtils.clamp(addOn + kanjiToLvl, 1, 6); **break**;  
 **case "kanjiTera"**: kanjiTeraLvl = MathUtils.clamp(addOn + kanjiTeraLvl, 1, 6); **break**;  
 **case "kanjiSoto"**: kanjiSotoLvl = MathUtils.clamp(addOn + kanjiSotoLvl, 1, 6); **break**;  
 **case "kanjiHi"**: kanjiHiLvl = MathUtils.clamp(addOn + kanjiHiLvl, 1, 6); **break**;  
 **case "kanjiAka"**: kanjiAkaLvl = MathUtils.clamp(addOn + kanjiAkaLvl, 1, 6); **break**;  
 **case "kanjiHana"**: kanjiHanaLvl = MathUtils.clamp(addOn + kanjiHanaLvl, 1, 6); **break**;  
 **case "kanjiMori"**: kanjiMoriLvl = MathUtils.clamp(addOn + kanjiMoriLvl, 1, 6); **break**;  
 **case "kanjiIshi"**: kanjiIshiLvl = MathUtils.clamp(addOn + kanjiIshiLvl, 1, 6); **break**;  
 **case "kanjiTsuchi"**: kanjiTsuchiLvl = MathUtils.clamp(addOn + kanjiTsuchiLvl, 1, 6); **break**;  
 **case "kanjiYama"**: kanjiYamaLvl = MathUtils.clamp(addOn + kanjiYamaLvl, 1, 6); **break**;  
 **case "kanjiSakana"**: kanjiSakanaLvl = MathUtils.clamp(addOn + kanjiSakanaLvl, 1, 6); **break**;  
 **case "kanjiKai"**: kanjiKaiLvl = MathUtils.clamp(addOn + kanjiKaiLvl, 1, 6); **break**;  
 **case "kanjiYuki"**: kanjiYukiLvl = MathUtils.clamp(addOn + kanjiYukiLvl, 1, 6); **break**;  
 **case "kanjiKawa"**: kanjiKawaLvl = MathUtils.clamp(addOn + kanjiKawaLvl, 1, 6); **break**;  
 **case "kanjiFuyu"**: kanjiFuyuLvl = MathUtils.clamp(addOn + kanjiFuyuLvl, 1, 6); **break**;  
 **case "kanjiMon"**: kanjiMonLvl = MathUtils.clamp(addOn + kanjiMonLvl, 1, 6); **break**;  
 **case "kanjiAna"**: kanjiAnaLvl = MathUtils.clamp(addOn + kanjiAnaLvl, 1, 6); **break**;  
 **case "kanjiKa"**: kanjiKaLvl = MathUtils.clamp(addOn + kanjiKaLvl, 1, 6); **break**;  
 **case "kanjiChi"**: kanjiChiLvl = MathUtils.clamp(addOn + kanjiChiLvl, 1, 6); **break**;  
 **case "kanjiHikari"**: kanjiHikariLvl = MathUtils.clamp(addOn + kanjiHikariLvl, 1, 6); **break**;  
 **default**: **break**;  
  
 }  
 }  
  
 **public static void** decreaseLvl (String letter, **int** takeOff){  
 **switch** (letter){  
 **case "hiraganaA"**: hiraganaALvl = MathUtils.clamp(takeOff - hiraganaALvl, 1, 4); **break**;  
 **case "hiraganaI"**: hiraganaILvl = MathUtils.clamp(takeOff - hiraganaILvl, 1, 4); **break**;  
 **case "hiraganaU"**: hiraganaULvl = MathUtils.clamp(takeOff - hiraganaULvl, 1, 4); **break**;  
 **case "hiraganaE"**: hiraganaELvl = MathUtils.clamp(takeOff - hiraganaELvl, 1, 4); **break**;  
 **case "hiraganaO"**: hiraganaOLvl = MathUtils.clamp(takeOff - hiraganaOLvl, 1, 4); **break**;  
 **case "hiraganaKa"**: hiraganaKaLvl = MathUtils.clamp(takeOff - hiraganaKaLvl, 1, 4); **break**;  
 **case "hiraganaKi"**: hiraganaKiLvl = MathUtils.clamp(takeOff - hiraganaKiLvl, 1, 4); **break**;  
 **case "hiraganaKu"**: hiraganaKuLvl = MathUtils.clamp(takeOff - hiraganaKuLvl, 1, 4); **break**;  
 **case "hiraganaKe"**: hiraganaKeLvl = MathUtils.clamp(takeOff - hiraganaKeLvl, 1, 4); **break**;  
 **case "hiraganaKo"**: hiraganaKoLvl = MathUtils.clamp(takeOff - hiraganaKoLvl, 1, 4); **break**;  
 **case "hiraganaSa"**: hiraganaSaLvl = MathUtils.clamp(takeOff - hiraganaSaLvl, 1, 4); **break**;  
 **case "hiraganaShi"**: hiraganaShiLvl = MathUtils.clamp(takeOff - hiraganaShiLvl, 1, 4); **break**;  
 **case "hiraganaSu"**: hiraganaSuLvl = MathUtils.clamp(takeOff - hiraganaSuLvl, 1, 4); **break**;  
 **case "hiraganaSe"**: hiraganaSeLvl = MathUtils.clamp(takeOff - hiraganaSeLvl, 1, 4); **break**;  
 **case "hiraganaSo"**: hiraganaSoLvl = MathUtils.clamp(takeOff - hiraganaSoLvl, 1, 4); **break**;  
 **case "hiraganaTa"**: hiraganaTaLvl = MathUtils.clamp(takeOff - hiraganaTaLvl, 1, 4); **break**;  
 **case "hiraganaChi"**: hiraganaChiLvl = MathUtils.clamp(takeOff - hiraganaChiLvl, 1, 4); **break**;  
 **case "hiraganaTsu"**: hiraganaTsuLvl = MathUtils.clamp(takeOff - hiraganaTsuLvl, 1, 4); **break**;  
 **case "hiraganaTe"**: hiraganaTeLvl = MathUtils.clamp(takeOff - hiraganaTeLvl, 1, 4); **break**;  
 **case "hiraganaTo"**: hiraganaToLvl = MathUtils.clamp(takeOff - hiraganaToLvl, 1, 4); **break**;  
 **case "hiraganaNa"**: hiraganaNaLvl = MathUtils.clamp(takeOff - hiraganaNaLvl, 1, 4); **break**;  
 **case "hiraganaNi"**: hiraganaNiLvl = MathUtils.clamp(takeOff - hiraganaNiLvl, 1, 4); **break**;  
 **case "hiraganaNu"**: hiraganaNuLvl = MathUtils.clamp(takeOff - hiraganaNuLvl, 1, 4); **break**;  
 **case "hiraganaNe"**: hiraganaNeLvl = MathUtils.clamp(takeOff - hiraganaNeLvl, 1, 4); **break**;  
 **case "hiraganaNo"**: hiraganaNoLvl = MathUtils.clamp(takeOff - hiraganaNoLvl, 1, 4); **break**;  
 **case "hiraganaHa"**: hiraganaHaLvl = MathUtils.clamp(takeOff - hiraganaHaLvl, 1, 4); **break**;  
 **case "hiraganaHi"**: hiraganaHiLvl = MathUtils.clamp(takeOff - hiraganaHiLvl, 1, 4); **break**;  
 **case "hiraganaHu"**: hiraganaHuLvl = MathUtils.clamp(takeOff - hiraganaHuLvl, 1, 4); **break**;  
 **case "hiraganaHe"**: hiraganaHeLvl = MathUtils.clamp(takeOff - hiraganaHeLvl, 1, 4); **break**;  
 **case "hiraganaHo"**: hiraganaHoLvl = MathUtils.clamp(takeOff - hiraganaHoLvl, 1, 4); **break**;  
 **case "hiraganaMa"**: hiraganaMaLvl = MathUtils.clamp(takeOff - hiraganaMaLvl, 1, 4); **break**;  
 **case "hiraganaMi"**: hiraganaMiLvl = MathUtils.clamp(takeOff - hiraganaMiLvl, 1, 4); **break**;  
 **case "hiraganaMu"**: hiraganaMuLvl = MathUtils.clamp(takeOff - hiraganaMuLvl, 1, 4); **break**;  
 **case "hiraganaMe"**: hiraganaMeLvl = MathUtils.clamp(takeOff - hiraganaMeLvl, 1, 4); **break**;  
 **case "hiraganaMo"**: hiraganaMoLvl = MathUtils.clamp(takeOff - hiraganaMoLvl, 1, 4); **break**;  
 **case "hiraganaRa"**: hiraganaRaLvl = MathUtils.clamp(takeOff - hiraganaRaLvl, 1, 4); **break**;  
 **case "hiraganaRi"**: hiraganaRiLvl = MathUtils.clamp(takeOff - hiraganaRiLvl, 1, 4); **break**;  
 **case "hiraganaRu"**: hiraganaRuLvl = MathUtils.clamp(takeOff - hiraganaRuLvl, 1, 4); **break**;  
 **case "hiraganaRe"**: hiraganaReLvl = MathUtils.clamp(takeOff - hiraganaReLvl, 1, 4); **break**;  
 **case "hiraganaRo"**: hiraganaRoLvl = MathUtils.clamp(takeOff - hiraganaRoLvl, 1, 4); **break**;  
 **case "hiraganaYa"**: hiraganaYaLvl = MathUtils.clamp(takeOff - hiraganaYaLvl, 1, 4); **break**;  
 **case "hiraganaYu"**: hiraganaYuLvl = MathUtils.clamp(takeOff - hiraganaYuLvl, 1, 4); **break**;  
 **case "hiraganaYo"**: hiraganaYoLvl = MathUtils.clamp(takeOff - hiraganaYoLvl, 1, 4); **break**;  
 **case "hiraganaWa"**: hiraganaWaLvl = MathUtils.clamp(takeOff - hiraganaWaLvl, 1, 4); **break**;  
 **case "hiraganaWo"**: hiraganaWoLvl = MathUtils.clamp(takeOff - hiraganaWoLvl, 1, 4); **break**;  
 **case "hiraganaN"**: hiraganaNLvl = MathUtils.clamp(takeOff - hiraganaNLvl, 1, 4); **break**;  
 **case "hiraganaGa"**: hiraganaGaLvl = MathUtils.clamp(takeOff - hiraganaGaLvl, 1, 4); **break**;  
 **case "hiraganaGi"**: hiraganaGiLvl = MathUtils.clamp(takeOff - hiraganaGiLvl, 1, 4); **break**;  
 **case "hiraganaGu"**: hiraganaGuLvl = MathUtils.clamp(takeOff - hiraganaGuLvl, 1, 4); **break**;  
 **case "hiraganaGe"**: hiraganaGeLvl = MathUtils.clamp(takeOff - hiraganaGeLvl, 1, 4); **break**;  
 **case "hiraganaGo"**: hiraganaGoLvl = MathUtils.clamp(takeOff - hiraganaGoLvl, 1, 4); **break**;  
 **case "hiraganaZa"**: hiraganaZaLvl = MathUtils.clamp(takeOff - hiraganaZaLvl, 1, 4); **break**;  
 **case "hiraganaJi"**: hiraganaJiLvl = MathUtils.clamp(takeOff - hiraganaJiLvl, 1, 4); **break**;  
 **case "hiraganaZu"**: hiraganaZuLvl = MathUtils.clamp(takeOff - hiraganaZuLvl, 1, 4); **break**;  
 **case "hiraganaZe"**: hiraganaZeLvl = MathUtils.clamp(takeOff - hiraganaZeLvl, 1, 4); **break**;  
 **case "hiraganaZo"**: hiraganaZoLvl = MathUtils.clamp(takeOff - hiraganaZoLvl, 1, 4); **break**;  
 **case "hiraganaDa"**: hiraganaDaLvl = MathUtils.clamp(takeOff - hiraganaDaLvl, 1, 4); **break**;  
 **case "hiraganaDi"**: hiraganaDiLvl = MathUtils.clamp(takeOff - hiraganaDiLvl, 1, 4); **break**;  
 **case "hiraganaDu"**: hiraganaDuLvl = MathUtils.clamp(takeOff - hiraganaDuLvl, 1, 4); **break**;  
 **case "hiraganaDe"**: hiraganaDeLvl = MathUtils.clamp(takeOff - hiraganaDeLvl, 1, 4); **break**;  
 **case "hiraganaDo"**: hiraganaDoLvl = MathUtils.clamp(takeOff - hiraganaDoLvl, 1, 4); **break**;  
 **case "hiraganaBa"**: hiraganaBaLvl = MathUtils.clamp(takeOff - hiraganaBaLvl, 1, 4); **break**;  
 **case "hiraganaBi"**: hiraganaBiLvl = MathUtils.clamp(takeOff - hiraganaBiLvl, 1, 4); **break**;  
 **case "hiraganaBu"**: hiraganaBuLvl = MathUtils.clamp(takeOff - hiraganaBuLvl, 1, 4); **break**;  
 **case "hiraganaBe"**: hiraganaBeLvl = MathUtils.clamp(takeOff - hiraganaBeLvl, 1, 4); **break**;  
 **case "hiraganaBo"**: hiraganaBoLvl = MathUtils.clamp(takeOff - hiraganaBoLvl, 1, 4); **break**;  
 **case "hiraganaPa"**: hiraganaPaLvl = MathUtils.clamp(takeOff - hiraganaPaLvl, 1, 4); **break**;  
 **case "hiraganaPi"**: hiraganaPiLvl = MathUtils.clamp(takeOff - hiraganaPiLvl, 1, 4); **break**;  
 **case "hiraganaPu"**: hiraganaPuLvl = MathUtils.clamp(takeOff - hiraganaPuLvl, 1, 4); **break**;  
 **case "hiraganaPe"**: hiraganaPeLvl = MathUtils.clamp(takeOff - hiraganaPeLvl, 1, 4); **break**;  
 **case "hiraganaPo"**: hiraganaPoLvl = MathUtils.clamp(takeOff - hiraganaPoLvl, 1, 4); **break**;  
 **case "hiraganaKya"**: hiraganaKyaLvl = MathUtils.clamp(takeOff - hiraganaKyaLvl, 1, 4); **break**;  
 **case "hiraganaKyu"**: hiraganaKyuLvl = MathUtils.clamp(takeOff - hiraganaKyuLvl, 1, 4); **break**;  
 **case "hiraganaKyo"**: hiraganaKyoLvl = MathUtils.clamp(takeOff - hiraganaKyoLvl, 1, 4); **break**;  
 **case "hiraganaSha"**: hiraganaShaLvl = MathUtils.clamp(takeOff - hiraganaShaLvl, 1, 4); **break**;  
 **case "hiraganaShu"**: hiraganaShuLvl = MathUtils.clamp(takeOff - hiraganaShuLvl, 1, 4); **break**;  
 **case "hiraganaSho"**: hiraganaShoLvl = MathUtils.clamp(takeOff - hiraganaShoLvl, 1, 4); **break**;  
 **case "hiraganaCha"**: hiraganaChaLvl = MathUtils.clamp(takeOff - hiraganaChaLvl, 1, 4); **break**;  
 **case "hiraganaChu"**: hiraganaChuLvl = MathUtils.clamp(takeOff - hiraganaChuLvl, 1, 4); **break**;  
 **case "hiraganaCho"**: hiraganaChoLvl = MathUtils.clamp(takeOff - hiraganaChoLvl, 1, 4); **break**;  
 **case "hiraganaHya"**: hiraganaHyaLvl = MathUtils.clamp(takeOff - hiraganaHyaLvl, 1, 4); **break**;  
 **case "hiraganaHyu"**: hiraganaHyuLvl = MathUtils.clamp(takeOff - hiraganaHyuLvl, 1, 4); **break**;  
 **case "hiraganaHyo"**: hiraganaHyoLvl = MathUtils.clamp(takeOff - hiraganaHyoLvl, 1, 4); **break**;  
 **case "hiraganaNya"**: hiraganaNyaLvl = MathUtils.clamp(takeOff - hiraganaNyaLvl, 1, 4); **break**;  
 **case "hiraganaNyu"**: hiraganaNyuLvl = MathUtils.clamp(takeOff - hiraganaNyuLvl, 1, 4); **break**;  
 **case "hiraganaNyo"**: hiraganaNyoLvl = MathUtils.clamp(takeOff - hiraganaNyoLvl, 1, 4); **break**;  
 **case "hiraganaMya"**: hiraganaMyaLvl = MathUtils.clamp(takeOff - hiraganaMyaLvl, 1, 4); **break**;  
 **case "hiraganaMyu"**: hiraganaMyuLvl = MathUtils.clamp(takeOff - hiraganaMyuLvl, 1, 4); **break**;  
 **case "hiraganaMyo"**: hiraganaMyoLvl = MathUtils.clamp(takeOff - hiraganaMyoLvl, 1, 4); **break**;  
 **case "hiraganaRya"**: hiraganaRyaLvl = MathUtils.clamp(takeOff - hiraganaRyaLvl, 1, 4); **break**;  
 **case "hiraganaRyu"**: hiraganaRyuLvl = MathUtils.clamp(takeOff - hiraganaRyuLvl, 1, 4); **break**;  
 **case "hiraganaRyo"**: hiraganaRyoLvl = MathUtils.clamp(takeOff - hiraganaRyoLvl, 1, 4); **break**;  
 **case "hiraganaGya"**: hiraganaGyaLvl = MathUtils.clamp(takeOff - hiraganaGyaLvl, 1, 4); **break**;  
 **case "hiraganaGyu"**: hiraganaGyuLvl = MathUtils.clamp(takeOff - hiraganaGyuLvl, 1, 4); **break**;  
 **case "hiraganaGyo"**: hiraganaGyoLvl = MathUtils.clamp(takeOff - hiraganaGyoLvl, 1, 4); **break**;  
 **case "hiraganaJya"**: hiraganaJyaLvl = MathUtils.clamp(takeOff - hiraganaJyaLvl, 1, 4); **break**;  
 **case "hiraganaJyu"**: hiraganaJyuLvl = MathUtils.clamp(takeOff - hiraganaJyuLvl, 1, 4); **break**;  
 **case "hiraganaJyo"**: hiraganaJyoLvl = MathUtils.clamp(takeOff - hiraganaJyoLvl, 1, 4); **break**;  
 **case "hiraganaDya"**: hiraganaDyaLvl = MathUtils.clamp(takeOff - hiraganaDyaLvl, 1, 4); **break**;  
 **case "hiraganaDyu"**: hiraganaDyuLvl = MathUtils.clamp(takeOff - hiraganaDyuLvl, 1, 4); **break**;  
 **case "hiraganaDyo"**: hiraganaDyoLvl = MathUtils.clamp(takeOff - hiraganaDyoLvl, 1, 4); **break**;  
 **case "hiraganaBya"**: hiraganaByaLvl = MathUtils.clamp(takeOff - hiraganaByaLvl, 1, 4); **break**;  
 **case "hiraganaByu"**: hiraganaByuLvl = MathUtils.clamp(takeOff - hiraganaByuLvl, 1, 4); **break**;  
 **case "hiraganaByo"**: hiraganaByoLvl = MathUtils.clamp(takeOff - hiraganaByoLvl, 1, 4); **break**;  
 **case "hiraganaPya"**: hiraganaPyaLvl = MathUtils.clamp(takeOff - hiraganaPyaLvl, 1, 4); **break**;  
 **case "hiraganaPyu"**: hiraganaPyuLvl = MathUtils.clamp(takeOff - hiraganaPyuLvl, 1, 4); **break**;  
 **case "hiraganaPyo"**: hiraganaPyoLvl = MathUtils.clamp(takeOff - hiraganaPyoLvl, 1, 4); **break**;  
 **case "katakanaA"**: katakanaALvl = MathUtils.clamp(takeOff - katakanaALvl, 1, 4); **break**;  
 **case "katakanaI"**: katakanaILvl = MathUtils.clamp(takeOff - katakanaILvl, 1, 4); **break**;  
 **case "katakanaU"**: katakanaULvl = MathUtils.clamp(takeOff - katakanaULvl, 1, 4); **break**;  
 **case "katakanaE"**: katakanaELvl = MathUtils.clamp(takeOff - katakanaELvl, 1, 4); **break**;  
 **case "katakanaO"**: katakanaOLvl = MathUtils.clamp(takeOff - katakanaOLvl, 1, 4); **break**;  
 **case "katakanaKa"**: katakanaKaLvl = MathUtils.clamp(takeOff - katakanaKaLvl, 1, 4); **break**;  
 **case "katakanaKi"**: katakanaKiLvl = MathUtils.clamp(takeOff - katakanaKiLvl, 1, 4); **break**;  
 **case "katakanaKu"**: katakanaKuLvl = MathUtils.clamp(takeOff - katakanaKuLvl, 1, 4); **break**;  
 **case "katakanaKe"**: katakanaKeLvl = MathUtils.clamp(takeOff - katakanaKeLvl, 1, 4); **break**;  
 **case "katakanaKo"**: katakanaKoLvl = MathUtils.clamp(takeOff - katakanaKoLvl, 1, 4); **break**;  
 **case "katakanaSa"**: katakanaSaLvl = MathUtils.clamp(takeOff - katakanaSaLvl, 1, 4); **break**;  
 **case "katakanaLvlShi"**: katakanaShiLvl = MathUtils.clamp(takeOff - katakanaShiLvl, 1, 4); **break**;  
 **case "katakanaLvlSu"**: katakanaSuLvl = MathUtils.clamp(takeOff - katakanaSuLvl, 1, 4); **break**;  
 **case "katakanaLvlSe"**: katakanaSeLvl = MathUtils.clamp(takeOff - katakanaSeLvl, 1, 4); **break**;  
 **case "katakanaLvlSo"**: katakanaSoLvl = MathUtils.clamp(takeOff - katakanaSoLvl, 1, 4); **break**;  
 **case "katakanaLvlTa"**: katakanaTaLvl = MathUtils.clamp(takeOff - katakanaTaLvl, 1, 4); **break**;  
 **case "katakanaLvlChi"**: katakanaChiLvl = MathUtils.clamp(takeOff - katakanaChiLvl, 1, 4); **break**;  
 **case "katakanaLvlTsu"**: katakanaTsuLvl = MathUtils.clamp(takeOff - katakanaTsuLvl, 1, 4); **break**;  
 **case "katakanaLvlTe"**: katakanaTeLvl = MathUtils.clamp(takeOff - katakanaTeLvl, 1, 4); **break**;  
 **case "katakanaLvlTo"**: katakanaToLvl = MathUtils.clamp(takeOff - katakanaToLvl, 1, 4); **break**;  
 **case "katakanaLvlNa"**: katakanaNaLvl = MathUtils.clamp(takeOff - katakanaNaLvl, 1, 4); **break**;  
 **case "katakanaLvlNi"**: katakanaNiLvl = MathUtils.clamp(takeOff - katakanaNiLvl, 1, 4); **break**;  
 **case "katakanaLvlNu"**: katakanaNuLvl = MathUtils.clamp(takeOff - katakanaNuLvl, 1, 4); **break**;  
 **case "katakanaLvlNe"**: katakanaNeLvl = MathUtils.clamp(takeOff - katakanaNeLvl, 1, 4); **break**;  
 **case "katakanaLvlNo"**: katakanaNoLvl = MathUtils.clamp(takeOff - katakanaNoLvl, 1, 4); **break**;  
 **case "katakanaLvlHa"**: katakanaHaLvl = MathUtils.clamp(takeOff - katakanaHaLvl, 1, 4); **break**;  
 **case "katakanaLvlHi"**: katakanaHiLvl = MathUtils.clamp(takeOff - katakanaHiLvl, 1, 4); **break**;  
 **case "katakanaLvlHu"**: katakanaHuLvl = MathUtils.clamp(takeOff - katakanaHuLvl, 1, 4); **break**;  
 **case "katakanaLvlHe"**: katakanaHeLvl = MathUtils.clamp(takeOff - katakanaHeLvl, 1, 4); **break**;  
 **case "katakanaLvlHo"**: katakanaHoLvl = MathUtils.clamp(takeOff - katakanaHoLvl, 1, 4); **break**;  
 **case "katakanaLvlMa"**: katakanaMaLvl = MathUtils.clamp(takeOff - katakanaMaLvl, 1, 4); **break**;  
 **case "katakanaLvlMi"**: katakanaMiLvl = MathUtils.clamp(takeOff - katakanaMiLvl, 1, 4); **break**;  
 **case "katakanaLvlMu"**: katakanaMuLvl = MathUtils.clamp(takeOff - katakanaMuLvl, 1, 4); **break**;  
 **case "katakanaLvlMe"**: katakanaMeLvl = MathUtils.clamp(takeOff - katakanaMeLvl, 1, 4); **break**;  
 **case "katakanaLvlMo"**: katakanaMoLvl = MathUtils.clamp(takeOff - katakanaMoLvl, 1, 4); **break**;  
 **case "katakanaLvlRa"**: katakanaRaLvl = MathUtils.clamp(takeOff - katakanaRaLvl, 1, 4); **break**;  
 **case "katakanaLvlRi"**: katakanaRiLvl = MathUtils.clamp(takeOff - katakanaRiLvl, 1, 4); **break**;  
 **case "katakanaLvlRu"**: katakanaRuLvl = MathUtils.clamp(takeOff - katakanaRuLvl, 1, 4); **break**;  
 **case "katakanaLvlRe"**: katakanaReLvl = MathUtils.clamp(takeOff - katakanaReLvl, 1, 4); **break**;  
 **case "katakanaLvlRo"**: katakanaRoLvl = MathUtils.clamp(takeOff - katakanaRoLvl, 1, 4); **break**;  
 **case "katakanaLvlYa"**: katakanaYaLvl = MathUtils.clamp(takeOff - katakanaYaLvl, 1, 4); **break**;  
 **case "katakanaLvlYu"**: katakanaYuLvl = MathUtils.clamp(takeOff - katakanaYuLvl, 1, 4); **break**;  
 **case "katakanaLvlYo"**: katakanaYoLvl = MathUtils.clamp(takeOff - katakanaYoLvl, 1, 4); **break**;  
 **case "katakanaLvlWa"**: katakanaWaLvl = MathUtils.clamp(takeOff - katakanaWaLvl, 1, 4); **break**;  
 **case "katakanaLvlWo"**: katakanaWoLvl = MathUtils.clamp(takeOff - katakanaWoLvl, 1, 4); **break**;  
 **case "katakanaLvlN"**: katakanaNLvl = MathUtils.clamp(takeOff - katakanaNLvl, 1, 4); **break**;  
 **case "katakanaLvlGa"**: katakanaGaLvl = MathUtils.clamp(takeOff - katakanaGaLvl, 1, 4); **break**;  
 **case "katakanaLvlGi"**: katakanaGiLvl = MathUtils.clamp(takeOff - katakanaGiLvl, 1, 4); **break**;  
 **case "katakanaLvlGu"**: katakanaGuLvl = MathUtils.clamp(takeOff - katakanaGuLvl, 1, 4); **break**;  
 **case "katakanaLvlGe"**: katakanaGeLvl = MathUtils.clamp(takeOff - katakanaGeLvl, 1, 4); **break**;  
 **case "katakanaLvlGo"**: katakanaGoLvl = MathUtils.clamp(takeOff - katakanaGoLvl, 1, 4); **break**;  
 **case "katakanaLvlZa"**: katakanaZaLvl = MathUtils.clamp(takeOff - katakanaZaLvl, 1, 4); **break**;  
 **case "katakanaLvlJi"**: katakanaJiLvl = MathUtils.clamp(takeOff - katakanaJiLvl, 1, 4); **break**;  
 **case "katakanaLvlZu"**: katakanaZuLvl = MathUtils.clamp(takeOff - katakanaZuLvl, 1, 4); **break**;  
 **case "katakanaLvlZe"**: katakanaZeLvl = MathUtils.clamp(takeOff - katakanaZeLvl, 1, 4); **break**;  
 **case "katakanaLvlZo"**: katakanaZoLvl = MathUtils.clamp(takeOff - katakanaZoLvl, 1, 4); **break**;  
 **case "katakanaLvlDa"**: katakanaDaLvl = MathUtils.clamp(takeOff - katakanaDaLvl, 1, 4); **break**;  
 **case "katakanaLvlDi"**: katakanaDiLvl = MathUtils.clamp(takeOff - katakanaDiLvl, 1, 4); **break**;  
 **case "katakanaLvlDu"**: katakanaDuLvl = MathUtils.clamp(takeOff - katakanaDuLvl, 1, 4); **break**;  
 **case "katakanaLvlDe"**: katakanaDeLvl = MathUtils.clamp(takeOff - katakanaDeLvl, 1, 4); **break**;  
 **case "katakanaLvlDo"**: katakanaDoLvl = MathUtils.clamp(takeOff - katakanaDoLvl, 1, 4); **break**;  
 **case "katakanaLvlBa"**: katakanaBaLvl = MathUtils.clamp(takeOff - katakanaBaLvl, 1, 4); **break**;  
 **case "katakanaLvlBi"**: katakanaBiLvl = MathUtils.clamp(takeOff - katakanaBiLvl, 1, 4); **break**;  
 **case "katakanaLvlBu"**: katakanaBuLvl = MathUtils.clamp(takeOff - katakanaBuLvl, 1, 4); **break**;  
 **case "katakanaLvlBe"**: katakanaBeLvl = MathUtils.clamp(takeOff - katakanaBeLvl, 1, 4); **break**;  
 **case "katakanaLvlBo"**: katakanaBoLvl = MathUtils.clamp(takeOff - katakanaBoLvl, 1, 4); **break**;  
 **case "katakanaLvlPa"**: katakanaPaLvl = MathUtils.clamp(takeOff - katakanaPaLvl, 1, 4); **break**;  
 **case "katakanaLvlPi"**: katakanaPiLvl = MathUtils.clamp(takeOff - katakanaPiLvl, 1, 4); **break**;  
 **case "katakanaLvlPu"**: katakanaPuLvl = MathUtils.clamp(takeOff - katakanaPuLvl, 1, 4); **break**;  
 **case "katakanaLvlPe"**: katakanaPeLvl = MathUtils.clamp(takeOff - katakanaPeLvl, 1, 4); **break**;  
 **case "katakanaLvlPo"**: katakanaPoLvl = MathUtils.clamp(takeOff - katakanaPoLvl, 1, 4); **break**;  
 **case "katakanaLvlKya"**: katakanaKyaLvl = MathUtils.clamp(takeOff - katakanaKyaLvl, 1, 4); **break**;  
 **case "katakanaLvlKyu"**: katakanaKyuLvl = MathUtils.clamp(takeOff - katakanaKyuLvl, 1, 4); **break**;  
 **case "katakanaLvlKyo"**: katakanaKyoLvl = MathUtils.clamp(takeOff - katakanaKyoLvl, 1, 4); **break**;  
 **case "katakanaLvlSha"**: katakanaShaLvl = MathUtils.clamp(takeOff - katakanaShaLvl, 1, 4); **break**;  
 **case "katakanaLvlShu"**: katakanaShuLvl = MathUtils.clamp(takeOff - katakanaShuLvl, 1, 4); **break**;  
 **case "katakanaLvlSho"**: katakanaShoLvl = MathUtils.clamp(takeOff - katakanaShoLvl, 1, 4); **break**;  
 **case "katakanaLvlCha"**: katakanaChaLvl = MathUtils.clamp(takeOff - katakanaChaLvl, 1, 4); **break**;  
 **case "katakanaLvlChu"**: katakanaChuLvl = MathUtils.clamp(takeOff - katakanaChuLvl, 1, 4); **break**;  
 **case "katakanaLvlCho"**: katakanaChoLvl = MathUtils.clamp(takeOff - katakanaChoLvl, 1, 4); **break**;  
 **case "katakanaLvlHya"**: katakanaHyaLvl = MathUtils.clamp(takeOff - katakanaHyaLvl, 1, 4); **break**;  
 **case "katakanaLvlHyu"**: katakanaHyuLvl = MathUtils.clamp(takeOff - katakanaHyuLvl, 1, 4); **break**;  
 **case "katakanaLvlHyo"**: katakanaHyoLvl = MathUtils.clamp(takeOff - katakanaHyoLvl, 1, 4); **break**;  
 **case "katakanaLvlNya"**: katakanaNyaLvl = MathUtils.clamp(takeOff - katakanaNyaLvl, 1, 4); **break**;  
 **case "katakanaLvlNyu"**: katakanaNyuLvl = MathUtils.clamp(takeOff - katakanaNyuLvl, 1, 4); **break**;  
 **case "katakanaLvlNyo"**: katakanaNyoLvl = MathUtils.clamp(takeOff - katakanaNyoLvl, 1, 4); **break**;  
 **case "katakanaLvlMya"**: katakanaMyaLvl = MathUtils.clamp(takeOff - katakanaMyaLvl, 1, 4); **break**;  
 **case "katakanaLvlMyu"**: katakanaMyuLvl = MathUtils.clamp(takeOff - katakanaMyuLvl, 1, 4); **break**;  
 **case "katakanaLvlMyo"**: katakanaMyoLvl = MathUtils.clamp(takeOff - katakanaMyoLvl, 1, 4); **break**;  
 **case "katakanaLvlRya"**: katakanaRyaLvl = MathUtils.clamp(takeOff - katakanaRyaLvl, 1, 4); **break**;  
 **case "katakanaLvlRyu"**: katakanaRyuLvl = MathUtils.clamp(takeOff - katakanaRyuLvl, 1, 4); **break**;  
 **case "katakanaLvlRyo"**: katakanaRyoLvl = MathUtils.clamp(takeOff - katakanaRyoLvl, 1, 4); **break**;  
 **case "katakanaLvlGya"**: katakanaGyaLvl = MathUtils.clamp(takeOff - katakanaGyaLvl, 1, 4); **break**;  
 **case "katakanaLvlGyu"**: katakanaGyuLvl = MathUtils.clamp(takeOff - katakanaGyuLvl, 1, 4); **break**;  
 **case "katakanaLvlGyo"**: katakanaGyoLvl = MathUtils.clamp(takeOff - katakanaGyoLvl, 1, 4); **break**;  
 **case "katakanaLvlJya"**: katakanaJyaLvl = MathUtils.clamp(takeOff - katakanaJyaLvl, 1, 4); **break**;  
 **case "katakanaLvlJyu"**: katakanaJyuLvl = MathUtils.clamp(takeOff - katakanaJyuLvl, 1, 4); **break**;  
 **case "katakanaLvlJyo"**: katakanaJyoLvl = MathUtils.clamp(takeOff - katakanaJyoLvl, 1, 4); **break**;  
 **case "katakanaLvlDya"**: katakanaDyaLvl = MathUtils.clamp(takeOff - katakanaDyaLvl, 1, 4); **break**;  
 **case "katakanaLvlDyu"**: katakanaDyuLvl = MathUtils.clamp(takeOff - katakanaDyuLvl, 1, 4); **break**;  
 **case "katakanaLvlDyo"**: katakanaDyoLvl = MathUtils.clamp(takeOff - katakanaDyoLvl, 1, 4); **break**;  
 **case "katakanaLvlBya"**: katakanaByaLvl = MathUtils.clamp(takeOff - katakanaByaLvl, 1, 4); **break**;  
 **case "katakanaLvlByu"**: katakanaByuLvl = MathUtils.clamp(takeOff - katakanaByuLvl, 1, 4); **break**;  
 **case "katakanaLvlByo"**: katakanaByoLvl = MathUtils.clamp(takeOff - katakanaByoLvl, 1, 4); **break**;  
 **case "katakanaLvlPya"**: katakanaPyaLvl = MathUtils.clamp(takeOff - katakanaPyaLvl, 1, 4); **break**;  
 **case "katakanaLvlPyu"**: katakanaPyuLvl = MathUtils.clamp(takeOff - katakanaPyuLvl, 1, 4); **break**;  
 **case "katakanaLvlPyo"**: katakanaPyoLvl = MathUtils.clamp(takeOff - katakanaPyoLvl, 1, 4); **break**;  
 **case "kanjiHa"**: kanjiHaLvl = MathUtils.clamp(takeOff - kanjiHaLvl, 1, 6); **break**;  
 **case "kanjiKi"**: kanjiKiLvl = MathUtils.clamp(takeOff - kanjiKiLvl, 1, 6); **break**;  
 **case "kanjiMushi"**: kanjiMushiLvl = MathUtils.clamp(takeOff - kanjiMushiLvl, 1, 6); **break**;  
 **case "kanjiKaze"**: kanjiKazeLvl = MathUtils.clamp(takeOff - kanjiKazeLvl, 1, 6); **break**;  
 **case "kanjiUchi"**: kanjiUchiLvl = MathUtils.clamp(takeOff - kanjiUchiLvl, 1, 6); **break**;  
 **case "kanjiUmi"**: kanjiUmiLvl = MathUtils.clamp(takeOff - kanjiUmiLvl, 1, 6); **break**;  
 **case "kanjiMizu"**: kanjiMizuLvl = MathUtils.clamp(takeOff - kanjiMizuLvl, 1, 6); **break**;  
 **case "kanjiIke"**: kanjiIkeLvl = MathUtils.clamp(takeOff - kanjiIkeLvl, 1, 6); **break**;  
 **case "kanjiKoori"**: kanjiKooriLvl = MathUtils.clamp(takeOff - kanjiKooriLvl, 1, 6); **break**;  
 **case "kanjiHayashi"**:kanjiHayashiLvl = MathUtils.clamp(takeOff - kanjiHayashiLvl, 1, 6); **break**;  
 **case "kanjiTo"**: kanjiToLvl = MathUtils.clamp(takeOff - kanjiToLvl, 1, 6); **break**;  
 **case "kanjiTera"**: kanjiTeraLvl = MathUtils.clamp(takeOff - kanjiTeraLvl, 1, 6); **break**;  
 **case "kanjiSoto"**: kanjiSotoLvl = MathUtils.clamp(takeOff - kanjiSotoLvl, 1, 6); **break**;  
 **case "kanjiHi"**: kanjiHiLvl = MathUtils.clamp(takeOff - kanjiHiLvl, 1, 6); **break**;  
 **case "kanjiAka"**: kanjiAkaLvl = MathUtils.clamp(takeOff - kanjiAkaLvl, 1, 6); **break**;  
 **case "kanjiHana"**: kanjiHanaLvl = MathUtils.clamp(takeOff - kanjiHanaLvl, 1, 6); **break**;  
 **case "kanjiMori"**: kanjiMoriLvl = MathUtils.clamp(takeOff - kanjiMoriLvl, 1, 6); **break**;  
 **case "kanjiIshi"**: kanjiIshiLvl = MathUtils.clamp(takeOff - kanjiIshiLvl, 1, 6); **break**;  
 **case "kanjiTsuchi"**: kanjiTsuchiLvl = MathUtils.clamp(takeOff - kanjiTsuchiLvl, 1, 6); **break**;  
 **case "kanjiYama"**: kanjiYamaLvl = MathUtils.clamp(takeOff - kanjiYamaLvl, 1, 6); **break**;  
 **case "kanjiSakana"**: kanjiSakanaLvl = MathUtils.clamp(takeOff - kanjiSakanaLvl, 1, 6); **break**;  
 **case "kanjiKai"**: kanjiKaiLvl = MathUtils.clamp(takeOff - kanjiKaiLvl, 1, 6); **break**;  
 **case "kanjiYuki"**: kanjiYukiLvl = MathUtils.clamp(takeOff - kanjiYukiLvl, 1, 6); **break**;  
 **case "kanjiKawa"**: kanjiKawaLvl = MathUtils.clamp(takeOff - kanjiKawaLvl, 1, 6); **break**;  
 **case "kanjiFuyu"**: kanjiFuyuLvl = MathUtils.clamp(takeOff - kanjiFuyuLvl, 1, 6); **break**;  
 **case "kanjiMon"**: kanjiMonLvl = MathUtils.clamp(takeOff - kanjiMonLvl, 1, 6); **break**;  
 **case "kanjiAna"**: kanjiAnaLvl = MathUtils.clamp(takeOff - kanjiAnaLvl, 1, 6); **break**;  
 **case "kanjiKa"**: kanjiKaLvl = MathUtils.clamp(takeOff - kanjiKaLvl, 1, 6); **break**;  
 **case "kanjiChi"**: kanjiChiLvl = MathUtils.clamp(takeOff - kanjiChiLvl, 1, 6); **break**;  
 **case "kanjiHikari"**: kanjiHikariLvl = MathUtils.clamp(takeOff - kanjiHikariLvl, 1, 6); **break**;  
 **default**: **break**;  
  
 }  
 }  
  
  
 **public static void** setAllHiraganaMemorised(**boolean** temp){  
 allHiraganaMemorised = temp;  
 }  
  
 **public static void** setAllKatakanaMemorised(**boolean** temp) {  
 allKatakanaMemorised = temp;  
 }  
  
 **public static void** setAllKanjiMemorised(**boolean** temp){  
 allKanjiMemorised = temp;  
 }  
  
 **public static boolean** isAllHiraganaMemorised() {  
 **return** allHiraganaMemorised;  
 }  
  
 **public static boolean** isAllKatakanaMemorised() {  
 **return** allKatakanaMemorised;  
 }  
  
 **public static boolean** isAllKanjiMemorised() {  
 **return** allKanjiMemorised;  
 }  
  
 **public static** Array<Integer> getHiraganaList() {  
 hiraganaList = **new** Array<Integer>();  
  
 hiraganaList.add(hiraganaALvl);  
 hiraganaList.add(hiraganaILvl);  
 hiraganaList.add(hiraganaULvl);  
 hiraganaList.add(hiraganaELvl);  
 hiraganaList.add(hiraganaOLvl);  
 hiraganaList.add(hiraganaKaLvl);  
 hiraganaList.add(hiraganaKiLvl);  
 hiraganaList.add(hiraganaKuLvl);  
 hiraganaList.add(hiraganaKeLvl);  
 hiraganaList.add(hiraganaKoLvl);  
 hiraganaList.add(hiraganaSaLvl);  
 hiraganaList.add(hiraganaShiLvl);  
 hiraganaList.add(hiraganaSuLvl);  
 hiraganaList.add(hiraganaSeLvl);  
 hiraganaList.add(hiraganaSoLvl);  
 hiraganaList.add(hiraganaTaLvl);  
 hiraganaList.add(hiraganaChiLvl);  
 hiraganaList.add(hiraganaTsuLvl);  
 hiraganaList.add(hiraganaTeLvl);  
 hiraganaList.add(hiraganaToLvl);  
 hiraganaList.add(hiraganaNaLvl);  
 hiraganaList.add(hiraganaNiLvl);  
 hiraganaList.add(hiraganaNuLvl);  
 hiraganaList.add(hiraganaNeLvl);  
 hiraganaList.add(hiraganaNoLvl);  
 hiraganaList.add(hiraganaHaLvl);  
 hiraganaList.add(hiraganaHiLvl);  
 hiraganaList.add(hiraganaHuLvl);  
 hiraganaList.add(hiraganaHeLvl);  
 hiraganaList.add(hiraganaHoLvl);  
 hiraganaList.add(hiraganaMaLvl);  
 hiraganaList.add(hiraganaMiLvl);  
 hiraganaList.add(hiraganaMuLvl);  
 hiraganaList.add(hiraganaMeLvl);  
 hiraganaList.add(hiraganaMoLvl);  
 hiraganaList.add(hiraganaRaLvl);  
 hiraganaList.add(hiraganaRiLvl);  
 hiraganaList.add(hiraganaRuLvl);  
 hiraganaList.add(hiraganaReLvl);  
 hiraganaList.add(hiraganaRoLvl);  
 hiraganaList.add(hiraganaYuLvl);  
 hiraganaList.add(hiraganaYoLvl);  
 hiraganaList.add(hiraganaWaLvl);  
 hiraganaList.add(hiraganaWoLvl);  
 hiraganaList.add(hiraganaNLvl);  
 hiraganaList.add(hiraganaGaLvl);  
 hiraganaList.add(hiraganaGiLvl);  
 hiraganaList.add(hiraganaGuLvl);  
 hiraganaList.add(hiraganaGeLvl);  
 hiraganaList.add(hiraganaGoLvl);  
 hiraganaList.add(hiraganaZaLvl);  
 hiraganaList.add(hiraganaJiLvl);  
 hiraganaList.add(hiraganaZuLvl);  
 hiraganaList.add(hiraganaZeLvl);  
 hiraganaList.add(hiraganaZoLvl);  
 hiraganaList.add(hiraganaDaLvl);  
 hiraganaList.add(hiraganaDiLvl);  
 hiraganaList.add(hiraganaDuLvl);  
 hiraganaList.add(hiraganaDeLvl);  
 hiraganaList.add(hiraganaDoLvl);  
 hiraganaList.add(hiraganaBaLvl);  
 hiraganaList.add(hiraganaBiLvl);  
 hiraganaList.add(hiraganaBuLvl);  
 hiraganaList.add(hiraganaBeLvl);  
 hiraganaList.add(hiraganaBoLvl);  
 hiraganaList.add(hiraganaPaLvl);  
 hiraganaList.add(hiraganaPiLvl);  
 hiraganaList.add(hiraganaPuLvl);  
 hiraganaList.add(hiraganaPeLvl);  
 hiraganaList.add(hiraganaPoLvl);  
 hiraganaList.add(hiraganaYaLvl);  
 hiraganaList.add(hiraganaKyaLvl);  
 hiraganaList.add(hiraganaKyuLvl);  
 hiraganaList.add(hiraganaKyoLvl);  
 hiraganaList.add(hiraganaShaLvl);  
 hiraganaList.add(hiraganaShuLvl);  
 hiraganaList.add(hiraganaShoLvl);  
 hiraganaList.add(hiraganaChaLvl);  
 hiraganaList.add(hiraganaChuLvl);  
 hiraganaList.add(hiraganaChoLvl);  
 hiraganaList.add(hiraganaHyaLvl);  
 hiraganaList.add(hiraganaHyuLvl);  
 hiraganaList.add(hiraganaHyoLvl);  
 hiraganaList.add(hiraganaNyaLvl);  
 hiraganaList.add(hiraganaNyuLvl);  
 hiraganaList.add(hiraganaNyoLvl);  
 hiraganaList.add(hiraganaMyaLvl);  
 hiraganaList.add(hiraganaMyuLvl);  
 hiraganaList.add(hiraganaMyoLvl);  
 hiraganaList.add(hiraganaRyaLvl);  
 hiraganaList.add(hiraganaRyuLvl);  
 hiraganaList.add(hiraganaRyoLvl);  
 hiraganaList.add(hiraganaGyaLvl);  
 hiraganaList.add(hiraganaGyuLvl);  
 hiraganaList.add(hiraganaGyoLvl);  
 hiraganaList.add(hiraganaJyaLvl);  
 hiraganaList.add(hiraganaJyuLvl);  
 hiraganaList.add(hiraganaJyoLvl);  
 hiraganaList.add(hiraganaDyaLvl);  
 hiraganaList.add(hiraganaDyuLvl);  
 hiraganaList.add(hiraganaDyoLvl);  
 hiraganaList.add(hiraganaByaLvl);  
 hiraganaList.add(hiraganaByuLvl);  
 hiraganaList.add(hiraganaByoLvl);  
 hiraganaList.add(hiraganaPyaLvl);  
 hiraganaList.add(hiraganaPyuLvl);  
 hiraganaList.add(hiraganaPyoLvl);  
  
 **return** hiraganaList;  
 }  
  
 **public static** Array<Integer> getKatakanaList() {  
 katakanaList = **new** Array<Integer>();  
  
 katakanaList.add(katakanaALvl);  
 katakanaList.add(katakanaILvl);  
 katakanaList.add(katakanaULvl);  
 katakanaList.add(katakanaELvl);  
 katakanaList.add(katakanaOLvl);  
 katakanaList.add(katakanaKaLvl);  
 katakanaList.add(katakanaKiLvl);  
 katakanaList.add(katakanaKuLvl);  
 katakanaList.add(katakanaKeLvl);  
 katakanaList.add(katakanaKoLvl);  
 katakanaList.add(katakanaSaLvl);  
 katakanaList.add(katakanaShiLvl);  
 katakanaList.add(katakanaSuLvl);  
 katakanaList.add(katakanaSeLvl);  
 katakanaList.add(katakanaSoLvl);  
 katakanaList.add(katakanaTaLvl);  
 katakanaList.add(katakanaChiLvl);  
 katakanaList.add(katakanaTsuLvl);  
 katakanaList.add(katakanaTeLvl);  
 katakanaList.add(katakanaToLvl);  
 katakanaList.add(katakanaNaLvl);  
 katakanaList.add(katakanaNiLvl);  
 katakanaList.add(katakanaNuLvl);  
 katakanaList.add(katakanaNeLvl);  
 katakanaList.add(katakanaNoLvl);  
 katakanaList.add(katakanaHaLvl);  
 katakanaList.add(katakanaHiLvl);  
 katakanaList.add(katakanaHuLvl);  
 katakanaList.add(katakanaHeLvl);  
 katakanaList.add(katakanaHoLvl);  
 katakanaList.add(katakanaMaLvl);  
 katakanaList.add(katakanaMiLvl);  
 katakanaList.add(katakanaMuLvl);  
 katakanaList.add(katakanaMeLvl);  
 katakanaList.add(katakanaMoLvl);  
 katakanaList.add(katakanaRaLvl);  
 katakanaList.add(katakanaRiLvl);  
 katakanaList.add(katakanaRuLvl);  
 katakanaList.add(katakanaReLvl);  
 katakanaList.add(katakanaRoLvl);  
 katakanaList.add(katakanaYuLvl);  
 katakanaList.add(katakanaYoLvl);  
 katakanaList.add(katakanaWaLvl);  
 katakanaList.add(katakanaWoLvl);  
 katakanaList.add(katakanaNLvl);  
 katakanaList.add(katakanaGaLvl);  
 katakanaList.add(katakanaGiLvl);  
 katakanaList.add(katakanaGuLvl);  
 katakanaList.add(katakanaGeLvl);  
 katakanaList.add(katakanaGoLvl);  
 katakanaList.add(katakanaZaLvl);  
 katakanaList.add(katakanaJiLvl);  
 katakanaList.add(katakanaZuLvl);  
 katakanaList.add(katakanaZeLvl);  
 katakanaList.add(katakanaZoLvl);  
 katakanaList.add(katakanaDaLvl);  
 katakanaList.add(katakanaDiLvl);  
 katakanaList.add(katakanaDuLvl);  
 katakanaList.add(katakanaDeLvl);  
 katakanaList.add(katakanaDoLvl);  
 katakanaList.add(katakanaBaLvl);  
 katakanaList.add(katakanaBiLvl);  
 katakanaList.add(katakanaBuLvl);  
 katakanaList.add(katakanaBeLvl);  
 katakanaList.add(katakanaBoLvl);  
 katakanaList.add(katakanaPaLvl);  
 katakanaList.add(katakanaPiLvl);  
 katakanaList.add(katakanaPuLvl);  
 katakanaList.add(katakanaPeLvl);  
 katakanaList.add(katakanaPoLvl);  
 katakanaList.add(katakanaYaLvl);  
 katakanaList.add(katakanaKyaLvl);  
 katakanaList.add(katakanaKyuLvl);  
 katakanaList.add(katakanaKyoLvl);  
 katakanaList.add(katakanaShaLvl);  
 katakanaList.add(katakanaShuLvl);  
 katakanaList.add(katakanaShoLvl);  
 katakanaList.add(katakanaChaLvl);  
 katakanaList.add(katakanaChuLvl);  
 katakanaList.add(katakanaChoLvl);  
 katakanaList.add(katakanaHyaLvl);  
 katakanaList.add(katakanaHyuLvl);  
 katakanaList.add(katakanaHyoLvl);  
 katakanaList.add(katakanaNyaLvl);  
 katakanaList.add(katakanaNyuLvl);  
 katakanaList.add(katakanaNyoLvl);  
 katakanaList.add(katakanaMyaLvl);  
 katakanaList.add(katakanaMyuLvl);  
 katakanaList.add(katakanaMyoLvl);  
 katakanaList.add(katakanaRyaLvl);  
 katakanaList.add(katakanaRyuLvl);  
 katakanaList.add(katakanaRyoLvl);  
 katakanaList.add(katakanaGyaLvl);  
 katakanaList.add(katakanaGyuLvl);  
 katakanaList.add(katakanaGyoLvl);  
 katakanaList.add(katakanaJyaLvl);  
 katakanaList.add(katakanaJyuLvl);  
 katakanaList.add(katakanaJyoLvl);  
 katakanaList.add(katakanaDyaLvl);  
 katakanaList.add(katakanaDyuLvl);  
 katakanaList.add(katakanaDyoLvl);  
 katakanaList.add(katakanaByaLvl);  
 katakanaList.add(katakanaByuLvl);  
 katakanaList.add(katakanaByoLvl);  
 katakanaList.add(katakanaPyaLvl);  
 katakanaList.add(katakanaPyuLvl);  
 katakanaList.add(katakanaPyoLvl);  
  
 **return** katakanaList;  
 }  
  
 **public static** Array<Integer> getKanjiList() {  
  
 kanjiList = **new** Array<Integer>();  
  
 kanjiList.add( kanjiHaLvl);  
 kanjiList.add( kanjiKiLvl);  
 kanjiList.add( kanjiMushiLvl);  
 kanjiList.add( kanjiKazeLvl);  
 kanjiList.add(kanjiTsuchiLvl);  
 kanjiList.add( kanjiUchiLvl);  
 kanjiList.add( kanjiUmiLvl);  
 kanjiList.add( kanjiMizuLvl);  
 kanjiList.add( kanjiIkeLvl);  
 kanjiList.add( kanjiKooriLvl);  
 kanjiList.add(kanjiHayashiLvl);  
 kanjiList.add( kanjiToLvl);  
 kanjiList.add( kanjiTeraLvl);  
 kanjiList.add( kanjiSotoLvl);  
 kanjiList.add( kanjiHiLvl);  
 kanjiList.add( kanjiAkaLvl);  
 kanjiList.add( kanjiHanaLvl);  
 kanjiList.add( kanjiMoriLvl);  
 kanjiList.add( kanjiIshiLvl);  
 kanjiList.add( kanjiYamaLvl);  
 kanjiList.add(kanjiSakanaLvl);  
 kanjiList.add( kanjiKaiLvl);  
 kanjiList.add( kanjiYukiLvl);  
 kanjiList.add( kanjiKawaLvl);  
 kanjiList.add( kanjiFuyuLvl);  
 kanjiList.add( kanjiMonLvl);  
 kanjiList.add( kanjiAnaLvl);  
 kanjiList.add( kanjiKaLvl);  
 kanjiList.add( kanjiChiLvl);  
 kanjiList.add(kanjiHikariLvl);  
  
  
 **return** kanjiList;  
 }  
  
 **public static void** adjustLvl(String letter, **int** addOn){  
 **switch** (letter){  
 **case "hiraganaA"**: hiraganaALvl = addOn; **break**;  
 **case "hiraganaI"**: hiraganaILvl = addOn; **break**;  
 **case "hiraganaU"**: hiraganaULvl = addOn; **break**;  
 **case "hiraganaE"**: hiraganaELvl = addOn; **break**;  
 **case "hiraganaO"**: hiraganaOLvl = addOn; **break**;  
 **case "hiraganaKa"**: hiraganaKaLvl = addOn; **break**;  
 **case "hiraganaKi"**: hiraganaKiLvl = addOn; **break**;  
 **case "hiraganaKu"**: hiraganaKuLvl = addOn; **break**;  
 **case "hiraganaKe"**: hiraganaKeLvl = addOn; **break**;  
 **case "hiraganaKo"**: hiraganaKoLvl = addOn; **break**;  
 **case "hiraganaSa"**: hiraganaSaLvl = addOn; **break**;  
 **case "hiraganaShi"**: hiraganaShiLvl = addOn; **break**;  
 **case "hiraganaSu"**: hiraganaSuLvl = addOn; **break**;  
 **case "hiraganaSe"**: hiraganaSeLvl = addOn; **break**;  
 **case "hiraganaSo"**: hiraganaSoLvl = addOn; **break**;  
 **case "hiraganaTa"**: hiraganaTaLvl = addOn; **break**;  
 **case "hiraganaChi"**: hiraganaChiLvl = addOn; **break**;  
 **case "hiraganaTsu"**: hiraganaTsuLvl = addOn; **break**;  
 **case "hiraganaTe"**: hiraganaTeLvl = addOn; **break**;  
 **case "hiraganaTo"**: hiraganaToLvl = addOn; **break**;  
 **case "hiraganaNa"**: hiraganaNaLvl = addOn; **break**;  
 **case "hiraganaNi"**: hiraganaNiLvl = addOn; **break**;  
 **case "hiraganaNu"**: hiraganaNuLvl = addOn; **break**;  
 **case "hiraganaNe"**: hiraganaNeLvl = addOn; **break**;  
 **case "hiraganaNo"**: hiraganaNoLvl = addOn; **break**;  
 **case "hiraganaHa"**: hiraganaHaLvl = addOn; **break**;  
 **case "hiraganaHi"**: hiraganaHiLvl = addOn; **break**;  
 **case "hiraganaHu"**: hiraganaHuLvl = addOn; **break**;  
 **case "hiraganaHe"**: hiraganaHeLvl = addOn; **break**;  
 **case "hiraganaHo"**: hiraganaHoLvl = addOn; **break**;  
 **case "hiraganaMa"**: hiraganaMaLvl = addOn; **break**;  
 **case "hiraganaMi"**: hiraganaMiLvl = addOn; **break**;  
 **case "hiraganaMu"**: hiraganaMuLvl = addOn; **break**;  
 **case "hiraganaMe"**: hiraganaMeLvl = addOn; **break**;  
 **case "hiraganaMo"**: hiraganaMoLvl = addOn; **break**;  
 **case "hiraganaRa"**: hiraganaRaLvl = addOn; **break**;  
 **case "hiraganaRi"**: hiraganaRiLvl = addOn; **break**;  
 **case "hiraganaRu"**: hiraganaRuLvl = addOn; **break**;  
 **case "hiraganaRe"**: hiraganaReLvl = addOn; **break**;  
 **case "hiraganaRo"**: hiraganaRoLvl = addOn; **break**;  
 **case "hiraganaYa"**: hiraganaYaLvl = addOn; **break**;  
 **case "hiraganaYu"**: hiraganaYuLvl = addOn; **break**;  
 **case "hiraganaYo"**: hiraganaYoLvl = addOn; **break**;  
 **case "hiraganaWa"**: hiraganaWaLvl = addOn; **break**;  
 **case "hiraganaWo"**: hiraganaWoLvl = addOn; **break**;  
 **case "hiraganaN"**: hiraganaNLvl = addOn; **break**;  
 **case "hiraganaGa"**: hiraganaGaLvl = addOn; **break**;  
 **case "hiraganaGi"**: hiraganaGiLvl = addOn; **break**;  
 **case "hiraganaGu"**: hiraganaGuLvl = addOn; **break**;  
 **case "hiraganaGe"**: hiraganaGeLvl = addOn; **break**;  
 **case "hiraganaGo"**: hiraganaGoLvl = addOn; **break**;  
 **case "hiraganaZa"**: hiraganaZaLvl = addOn; **break**;  
 **case "hiraganaJi"**: hiraganaJiLvl = addOn; **break**;  
 **case "hiraganaZu"**: hiraganaZuLvl = addOn; **break**;  
 **case "hiraganaZe"**: hiraganaZeLvl = addOn; **break**;  
 **case "hiraganaZo"**: hiraganaZoLvl = addOn; **break**;  
 **case "hiraganaDa"**: hiraganaDaLvl = addOn; **break**;  
 **case "hiraganaDi"**: hiraganaDiLvl = addOn; **break**;  
 **case "hiraganaDu"**: hiraganaDuLvl = addOn; **break**;  
 **case "hiraganaDe"**: hiraganaDeLvl = addOn; **break**;  
 **case "hiraganaDo"**: hiraganaDoLvl = addOn; **break**;  
 **case "hiraganaBa"**: hiraganaBaLvl = addOn; **break**;  
 **case "hiraganaBi"**: hiraganaBiLvl = addOn; **break**;  
 **case "hiraganaBu"**: hiraganaBuLvl = addOn; **break**;  
 **case "hiraganaBe"**: hiraganaBeLvl = addOn; **break**;  
 **case "hiraganaBo"**: hiraganaBoLvl = addOn; **break**;  
 **case "hiraganaPa"**: hiraganaPaLvl = addOn; **break**;  
 **case "hiraganaPi"**: hiraganaPiLvl = addOn; **break**;  
 **case "hiraganaPu"**: hiraganaPuLvl = addOn; **break**;  
 **case "hiraganaPe"**: hiraganaPeLvl = addOn; **break**;  
 **case "hiraganaPo"**: hiraganaPoLvl = addOn; **break**;  
 **case "hiraganaKya"**: hiraganaKyaLvl = addOn; **break**;  
 **case "hiraganaKyu"**: hiraganaKyuLvl = addOn; **break**;  
 **case "hiraganaKyo"**: hiraganaKyoLvl = addOn; **break**;  
 **case "hiraganaSha"**: hiraganaShaLvl = addOn; **break**;  
 **case "hiraganaShu"**: hiraganaShuLvl = addOn; **break**;  
 **case "hiraganaSho"**: hiraganaShoLvl = addOn; **break**;  
 **case "hiraganaCha"**: hiraganaChaLvl = addOn; **break**;  
 **case "hiraganaChu"**: hiraganaChuLvl = addOn; **break**;  
 **case "hiraganaCho"**: hiraganaChoLvl = addOn; **break**;  
 **case "hiraganaHya"**: hiraganaHyaLvl = addOn; **break**;  
 **case "hiraganaHyu"**: hiraganaHyuLvl = addOn; **break**;  
 **case "hiraganaHyo"**: hiraganaHyoLvl = addOn; **break**;  
 **case "hiraganaNya"**: hiraganaNyaLvl = addOn; **break**;  
 **case "hiraganaNyu"**: hiraganaNyuLvl = addOn; **break**;  
 **case "hiraganaNyo"**: hiraganaNyoLvl = addOn; **break**;  
 **case "hiraganaMya"**: hiraganaMyaLvl = addOn; **break**;  
 **case "hiraganaMyu"**: hiraganaMyuLvl = addOn; **break**;  
 **case "hiraganaMyo"**: hiraganaMyoLvl = addOn; **break**;  
 **case "hiraganaRya"**: hiraganaRyaLvl = addOn; **break**;  
 **case "hiraganaRyu"**: hiraganaRyuLvl = addOn; **break**;  
 **case "hiraganaRyo"**: hiraganaRyoLvl = addOn; **break**;  
 **case "hiraganaGya"**: hiraganaGyaLvl = addOn; **break**;  
 **case "hiraganaGyu"**: hiraganaGyuLvl = addOn; **break**;  
 **case "hiraganaGyo"**: hiraganaGyoLvl = addOn; **break**;  
 **case "hiraganaJya"**: hiraganaJyaLvl = addOn; **break**;  
 **case "hiraganaJyu"**: hiraganaJyuLvl = addOn; **break**;  
 **case "hiraganaJyo"**: hiraganaJyoLvl = addOn; **break**;  
 **case "hiraganaDya"**: hiraganaDyaLvl = addOn; **break**;  
 **case "hiraganaDyu"**: hiraganaDyuLvl = addOn; **break**;  
 **case "hiraganaDyo"**: hiraganaDyoLvl = addOn; **break**;  
 **case "hiraganaBya"**: hiraganaByaLvl = addOn; **break**;  
 **case "hiraganaByu"**: hiraganaByuLvl = addOn; **break**;  
 **case "hiraganaByo"**: hiraganaByoLvl = addOn; **break**;  
 **case "hiraganaPya"**: hiraganaPyaLvl = addOn; **break**;  
 **case "hiraganaPyu"**: hiraganaPyuLvl = addOn; **break**;  
 **case "hiraganaPyo"**: hiraganaPyoLvl = addOn; **break**;  
 **case "katakanaA"**: katakanaALvl = addOn; **break**;  
 **case "katakanaI"**: katakanaILvl = addOn; **break**;  
 **case "katakanaU"**: katakanaULvl = addOn; **break**;  
 **case "katakanaE"**: katakanaELvl = addOn; **break**;  
 **case "katakanaO"**: katakanaOLvl = addOn; **break**;  
 **case "katakanaKa"**: katakanaKaLvl = addOn; **break**;  
 **case "katakanaKi"**: katakanaKiLvl = addOn; **break**;  
 **case "katakanaKu"**: katakanaKuLvl = addOn; **break**;  
 **case "katakanaKe"**: katakanaKeLvl = addOn; **break**;  
 **case "katakanaKo"**: katakanaKoLvl = addOn; **break**;  
 **case "katakanaSa"**: katakanaSaLvl = addOn; **break**;  
 **case "katakanaLvlShi"**: katakanaShiLvl = addOn; **break**;  
 **case "katakanaLvlSu"**: katakanaSuLvl = addOn; **break**;  
 **case "katakanaLvlSe"**: katakanaSeLvl = addOn; **break**;  
 **case "katakanaLvlSo"**: katakanaSoLvl = addOn; **break**;  
 **case "katakanaLvlTa"**: katakanaTaLvl = addOn; **break**;  
 **case "katakanaLvlChi"**: katakanaChiLvl = addOn; **break**;  
 **case "katakanaLvlTsu"**: katakanaTsuLvl = addOn; **break**;  
 **case "katakanaLvlTe"**: katakanaTeLvl = addOn; **break**;  
 **case "katakanaLvlTo"**: katakanaToLvl = addOn; **break**;  
 **case "katakanaLvlNa"**: katakanaNaLvl = addOn; **break**;  
 **case "katakanaLvlNi"**: katakanaNiLvl = addOn; **break**;  
 **case "katakanaLvlNu"**: katakanaNuLvl = addOn; **break**;  
 **case "katakanaLvlNe"**: katakanaNeLvl = addOn; **break**;  
 **case "katakanaLvlNo"**: katakanaNoLvl = addOn; **break**;  
 **case "katakanaLvlHa"**: katakanaHaLvl = addOn; **break**;  
 **case "katakanaLvlHi"**: katakanaHiLvl = addOn; **break**;  
 **case "katakanaLvlHu"**: katakanaHuLvl = addOn; **break**;  
 **case "katakanaLvlHe"**: katakanaHeLvl = addOn; **break**;  
 **case "katakanaLvlHo"**: katakanaHoLvl = addOn; **break**;  
 **case "katakanaLvlMa"**: katakanaMaLvl = addOn; **break**;  
 **case "katakanaLvlMi"**: katakanaMiLvl = addOn; **break**;  
 **case "katakanaLvlMu"**: katakanaMuLvl = addOn; **break**;  
 **case "katakanaLvlMe"**: katakanaMeLvl = addOn; **break**;  
 **case "katakanaLvlMo"**: katakanaMoLvl = addOn; **break**;  
 **case "katakanaLvlRa"**: katakanaRaLvl = addOn; **break**;  
 **case "katakanaLvlRi"**: katakanaRiLvl = addOn; **break**;  
 **case "katakanaLvlRu"**: katakanaRuLvl = addOn; **break**;  
 **case "katakanaLvlRe"**: katakanaReLvl = addOn; **break**;  
 **case "katakanaLvlRo"**: katakanaRoLvl = addOn; **break**;  
 **case "katakanaLvlYa"**: katakanaYaLvl = addOn; **break**;  
 **case "katakanaLvlYu"**: katakanaYuLvl = addOn; **break**;  
 **case "katakanaLvlYo"**: katakanaYoLvl = addOn; **break**;  
 **case "katakanaLvlWa"**: katakanaWaLvl = addOn; **break**;  
 **case "katakanaLvlWo"**: katakanaWoLvl = addOn; **break**;  
 **case "katakanaLvlN"**: katakanaNLvl = addOn; **break**;  
 **case "katakanaLvlGa"**: katakanaGaLvl = addOn; **break**;  
 **case "katakanaLvlGi"**: katakanaGiLvl = addOn; **break**;  
 **case "katakanaLvlGu"**: katakanaGuLvl = addOn; **break**;  
 **case "katakanaLvlGe"**: katakanaGeLvl = addOn; **break**;  
 **case "katakanaLvlGo"**: katakanaGoLvl = addOn; **break**;  
 **case "katakanaLvlZa"**: katakanaZaLvl = addOn; **break**;  
 **case "katakanaLvlJi"**: katakanaJiLvl = addOn; **break**;  
 **case "katakanaLvlZu"**: katakanaZuLvl = addOn; **break**;  
 **case "katakanaLvlZe"**: katakanaZeLvl = addOn; **break**;  
 **case "katakanaLvlZo"**: katakanaZoLvl = addOn; **break**;  
 **case "katakanaLvlDa"**: katakanaDaLvl = addOn; **break**;  
 **case "katakanaLvlDi"**: katakanaDiLvl = addOn; **break**;  
 **case "katakanaLvlDu"**: katakanaDuLvl = addOn; **break**;  
 **case "katakanaLvlDe"**: katakanaDeLvl = addOn; **break**;  
 **case "katakanaLvlDo"**: katakanaDoLvl = addOn; **break**;  
 **case "katakanaLvlBa"**: katakanaBaLvl = addOn; **break**;  
 **case "katakanaLvlBi"**: katakanaBiLvl = addOn; **break**;  
 **case "katakanaLvlBu"**: katakanaBuLvl = addOn; **break**;  
 **case "katakanaLvlBe"**: katakanaBeLvl = addOn; **break**;  
 **case "katakanaLvlBo"**: katakanaBoLvl = addOn; **break**;  
 **case "katakanaLvlPa"**: katakanaPaLvl = addOn; **break**;  
 **case "katakanaLvlPi"**: katakanaPiLvl = addOn; **break**;  
 **case "katakanaLvlPu"**: katakanaPuLvl = addOn; **break**;  
 **case "katakanaLvlPe"**: katakanaPeLvl = addOn; **break**;  
 **case "katakanaLvlPo"**: katakanaPoLvl = addOn; **break**;  
 **case "katakanaLvlKya"**: katakanaKyaLvl = addOn; **break**;  
 **case "katakanaLvlKyu"**: katakanaKyuLvl = addOn; **break**;  
 **case "katakanaLvlKyo"**: katakanaKyoLvl = addOn; **break**;  
 **case "katakanaLvlSha"**: katakanaShaLvl = addOn; **break**;  
 **case "katakanaLvlShu"**: katakanaShuLvl = addOn; **break**;  
 **case "katakanaLvlSho"**: katakanaShoLvl = addOn; **break**;  
 **case "katakanaLvlCha"**: katakanaChaLvl = addOn; **break**;  
 **case "katakanaLvlChu"**: katakanaChuLvl = addOn; **break**;  
 **case "katakanaLvlCho"**: katakanaChoLvl = addOn; **break**;  
 **case "katakanaLvlHya"**: katakanaHyaLvl = addOn; **break**;  
 **case "katakanaLvlHyu"**: katakanaHyuLvl = addOn; **break**;  
 **case "katakanaLvlHyo"**: katakanaHyoLvl = addOn; **break**;  
 **case "katakanaLvlNya"**: katakanaNyaLvl = addOn; **break**;  
 **case "katakanaLvlNyu"**: katakanaNyuLvl = addOn; **break**;  
 **case "katakanaLvlNyo"**: katakanaNyoLvl = addOn; **break**;  
 **case "katakanaLvlMya"**: katakanaMyaLvl = addOn; **break**;  
 **case "katakanaLvlMyu"**: katakanaMyuLvl = addOn; **break**;  
 **case "katakanaLvlMyo"**: katakanaMyoLvl = addOn; **break**;  
 **case "katakanaLvlRya"**: katakanaRyaLvl = addOn; **break**;  
 **case "katakanaLvlRyu"**: katakanaRyuLvl = addOn; **break**;  
 **case "katakanaLvlRyo"**: katakanaRyoLvl = addOn; **break**;  
 **case "katakanaLvlGya"**: katakanaGyaLvl = addOn; **break**;  
 **case "katakanaLvlGyu"**: katakanaGyuLvl = addOn; **break**;  
 **case "katakanaLvlGyo"**: katakanaGyoLvl = addOn; **break**;  
 **case "katakanaLvlJya"**: katakanaJyaLvl = addOn; **break**;  
 **case "katakanaLvlJyu"**: katakanaJyuLvl = addOn; **break**;  
 **case "katakanaLvlJyo"**: katakanaJyoLvl = addOn; **break**;  
 **case "katakanaLvlDya"**: katakanaDyaLvl = addOn; **break**;  
 **case "katakanaLvlDyu"**: katakanaDyuLvl = addOn; **break**;  
 **case "katakanaLvlDyo"**: katakanaDyoLvl = addOn; **break**;  
 **case "katakanaLvlBya"**: katakanaByaLvl = addOn; **break**;  
 **case "katakanaLvlByu"**: katakanaByuLvl = addOn; **break**;  
 **case "katakanaLvlByo"**: katakanaByoLvl = addOn; **break**;  
 **case "katakanaLvlPya"**: katakanaPyaLvl = addOn; **break**;  
 **case "katakanaLvlPyu"**: katakanaPyuLvl = addOn; **break**;  
 **case "katakanaLvlPyo"**: katakanaPyoLvl = addOn; **break**;  
 **case "kanjiHa"**: kanjiHaLvl = addOn; **break**;  
 **case "kanjiHana"**: kanjiKiLvl = addOn; **break**;  
 **case "kanjiKi"**: kanjiMushiLvl = addOn; **break**;  
 **case "kanjiMori"**: kanjiKazeLvl = addOn; **break**;  
 **case "kanjiMushi"**: kanjiKazeLvl = addOn; **break**;  
 **case "kanjiIshi"**: kanjiUchiLvl = addOn; **break**;  
 **case "kanjiKaze"**: kanjiUmiLvl = addOn; **break**;  
 **case "kanjiTsuchi"**: kanjiMizuLvl = addOn; **break**;  
 **case "kanjiUchi"**: kanjiIkeLvl = addOn; **break**;  
 **case "kanjiYama"**: kanjiKooriLvl = addOn; **break**;  
 **case "kanjiUmi"**: kanjiHayashiLvl = addOn; **break**;  
 **case "kanjiSakana"**: kanjiToLvl = addOn; **break**;  
 **case "kanjiMizu"**: kanjiTeraLvl = addOn; **break**;  
 **case "kanjiKai"**: kanjiSotoLvl = addOn; **break**;  
 **case "kanjiIke"**: kanjiHiLvl = addOn; **break**;  
 **case "kanjiYuki"**: kanjiAkaLvl = addOn; **break**;  
 **case "kanjiKoori"**: kanjiHanaLvl = addOn; **break**;  
 **case "kanjiKawa"**: kanjiMoriLvl = addOn; **break**;  
 **case "kanjiHayashi"**: kanjiIshiLvl = addOn; **break**;  
 **case "kanjiFuyu"**: kanjiTsuchiLvl = addOn; **break**;  
 **case "kanjiTo"**: kanjiYamaLvl = addOn; **break**;  
 **case "kanjiMon"**: kanjiSakanaLvl = addOn; **break**;  
 **case "kanjiTera"**: kanjiKaiLvl = addOn; **break**;  
 **case "kanjiAna"**: kanjiYukiLvl = addOn; **break**;  
 **case "kanjiSoto"**: kanjiKawaLvl = addOn; **break**;  
 **case "kanjiKa"**: kanjiFuyuLvl = addOn; **break**;  
 **case "kanjiHi"**: kanjiMonLvl = addOn; **break**;  
 **case "kanjiChi"**: kanjiAnaLvl = addOn; **break**;  
 **case "kanjiAka"**: kanjiKaLvl = addOn; **break**;  
 **case "kanjiHikari"**: kanjiChiLvl = addOn; **break**;  
 **default**: **break**;  
  
 }  
 }  
}

## src.com.mygdx.game.maps

**package** com.mygdx.game.maps;  
  
**import** com.badlogic.gdx.graphics.g2d.Batch;  
**import** com.badlogic.gdx.maps.MapProperties;  
**import** com.mygdx.game.tools.Entity;  
  
**public class** ArmorerMap **extends** Map {  
 **private static** String *mapPath* = **"tilemaps/armorer.tmx"**;  
 **private static** String *blacksmith* = **"json\_scripts/chara/blacksmith.json"**;  
  
 **public** ArmorerMap(){  
 **super**(MapFactory.MapType.***ARMORER***, *mapPath*);  
 **mapEntities**.add(initSpecialEntity(Entity.*getEntityConfig*(*blacksmith*)));  
 }  
  
 @Override  
 **public void** updateMapEntities(MapManager mapMgr, Batch batch, **float** delta){  
 **for**( Entity entity : **mapEntities**){  
 entity.update(mapMgr, batch, delta);  
 }  
 }  
  
 @Override  
 **public void** unloadMusic() {  
  
 }  
  
 @Override  
 **public void** loadMusic() {  
  
 }  
  
}

**package** com.mygdx.game.maps;  
  
**import** com.badlogic.gdx.graphics.g2d.Batch;  
**import** com.mygdx.game.audio.AudioObserver;  
**import** com.mygdx.game.tools.Entity;  
  
**public class** CaveMap **extends** Map {  
 **private static** String *mapPath* = **"tilemaps/cave.tmx"**;  
  
 **public** CaveMap(){  
 **super**(MapFactory.MapType.***CAVE***, *mapPath*);  
 }  
  
 @Override  
 **public void** updateMapEntities(MapManager mapMgr, Batch batch, **float** delta){  
 **for**( Entity entity : **mapEntities**){  
 entity.update(mapMgr, batch, delta);  
 }  
 }  
  
 @Override  
 **public void** unloadMusic() {  
 notify(AudioObserver.AudioCommand.***MUSIC\_STOP***, AudioObserver.AudioTypeEvent.***MUSIC\_CAVE***);  
 }  
  
 @Override  
 **public void** loadMusic() {  
 notify(AudioObserver.AudioCommand.***MUSIC\_LOAD***, AudioObserver.AudioTypeEvent.***MUSIC\_CAVE***);  
 notify(AudioObserver.AudioCommand.***MUSIC\_PLAY\_LOOP***, AudioObserver.AudioTypeEvent.***MUSIC\_CAVE***);  
 }  
  
}

**package** com.mygdx.game.maps;  
  
**import** com.badlogic.gdx.graphics.g2d.Batch;  
**import** com.mygdx.game.audio.AudioObserver;  
**import** com.mygdx.game.tools.Entity;  
  
**public class** CoastMap **extends** Map {  
 **private static** String *mapPath* = **"tilemaps/coast.tmx"**;  
  
 **public** CoastMap(){  
 **super**(MapFactory.MapType.***COAST***, *mapPath*);  
 }  
  
 @Override  
 **public void** updateMapEntities(MapManager mapMgr, Batch batch, **float** delta){  
 **for**( Entity entity : **mapEntities**){  
 entity.update(mapMgr, batch, delta);  
 }  
 }  
  
 @Override  
 **public void** unloadMusic() {  
 notify(AudioObserver.AudioCommand.***MUSIC\_STOP***, AudioObserver.AudioTypeEvent.***MUSIC\_COAST***);  
 }  
  
 @Override  
 **public void** loadMusic() {  
 notify(AudioObserver.AudioCommand.***MUSIC\_LOAD***, AudioObserver.AudioTypeEvent.***MUSIC\_COAST***);  
 notify(AudioObserver.AudioCommand.***MUSIC\_PLAY\_LOOP***, AudioObserver.AudioTypeEvent.***MUSIC\_COAST***);  
 }  
  
}

**package** com.mygdx.game.maps;  
  
**import** com.badlogic.gdx.graphics.g2d.Batch;  
**import** com.mygdx.game.audio.AudioObserver;  
**import** com.mygdx.game.tools.Entity;  
  
**public class** DesertTempleMap **extends** Map {  
 **private static** String *mapPath* = **"tilemaps/desert\_temple.tmx"**;  
  
 **public** DesertTempleMap(){  
 **super**(MapFactory.MapType.***DESERT\_TEMPLE***, *mapPath*);  
 }  
  
 @Override  
 **public void** updateMapEntities(MapManager mapMgr, Batch batch, **float** delta){  
 **for**( Entity entity : **mapEntities**){  
 entity.update(mapMgr, batch, delta);  
 }  
 }  
  
 @Override  
 **public void** unloadMusic() {  
 notify(AudioObserver.AudioCommand.***MUSIC\_STOP***, AudioObserver.AudioTypeEvent.***MUSIC\_DESERT\_TEMPLE***);  
 }  
  
 @Override  
 **public void** loadMusic() {  
 notify(AudioObserver.AudioCommand.***MUSIC\_LOAD***, AudioObserver.AudioTypeEvent.***MUSIC\_DESERT\_TEMPLE***);  
 notify(AudioObserver.AudioCommand.***MUSIC\_PLAY\_LOOP***, AudioObserver.AudioTypeEvent.***MUSIC\_DESERT\_TEMPLE***);  
 }  
}

**package** com.mygdx.game.maps;  
  
**import** com.badlogic.gdx.graphics.g2d.Batch;  
**import** com.mygdx.game.audio.AudioObserver;  
**import** com.mygdx.game.tools.Entity;  
  
**public class** ForestMap **extends** Map {  
 **private static** String *mapPath* = **"tilemaps/forest.tmx"**;  
  
 **public** ForestMap(){  
 **super**(MapFactory.MapType.***FOREST***, *mapPath*);  
 }  
  
 @Override  
 **public void** updateMapEntities(MapManager mapMgr, Batch batch, **float** delta){  
 **for**( Entity entity : **mapEntities**){  
 entity.update(mapMgr, batch, delta);  
 }  
 }  
  
 @Override  
 **public void** unloadMusic() {  
 notify(AudioObserver.AudioCommand.***MUSIC\_STOP***, AudioObserver.AudioTypeEvent.***MUSIC\_FOREST***);  
 }  
  
 @Override  
 **public void** loadMusic() {  
 notify(AudioObserver.AudioCommand.***MUSIC\_LOAD***, AudioObserver.AudioTypeEvent.***MUSIC\_FOREST***);  
 notify(AudioObserver.AudioCommand.***MUSIC\_PLAY\_LOOP***, AudioObserver.AudioTypeEvent.***MUSIC\_FOREST***);  
 }  
  
}

**package** com.mygdx.game.maps;  
  
**import** com.badlogic.gdx.graphics.g2d.Batch;  
**import** com.mygdx.game.tools.Entity;  
  
**public class** HerbShopMap **extends** Map {  
 **private static** String *mapPath* = **"tilemaps/herb\_shop.tmx"**;  
 **private static** String *herbalist* = **"json\_scripts/chara/herbalist.json"**;  
  
 HerbShopMap(){  
 **super**(MapFactory.MapType.***HERB\_SHOP***, *mapPath*);  
 **mapEntities**.add(initSpecialEntity(Entity.*getEntityConfig*(*herbalist*)));  
 }  
  
 @Override  
 **public void** updateMapEntities(MapManager mapMgr, Batch batch, **float** delta){  
 **for**( Entity entity : **mapEntities**){  
 entity.update(mapMgr, batch, delta);  
 }  
 }  
  
 @Override  
 **public void** unloadMusic() {  
  
 }  
  
 @Override  
 **public void** loadMusic() {  
  
 }  
  
}

**package** com.mygdx.game.maps;  
  
**import** com.badlogic.gdx.graphics.g2d.Batch;  
**import** com.mygdx.game.tools.Entity;  
  
**public class** HouseOneMap **extends** Map {  
 **private static** String *mapPath* = **"tilemaps/house\_one.tmx"**;  
 **private static** String *librarian* = **"json\_scripts/chara/librarian.json"**;  
 **private static** String *walkaround\_four* = **"json\_scripts/chara/walkaround\_four.json"**;  
  
 HouseOneMap(){  
 **super**(MapFactory.MapType.***HOUSE\_ONE***, *mapPath*);  
 **mapEntities**.add(initSpecialEntity(Entity.*getEntityConfig*(*librarian*)));  
 **mapEntities**.add(initSpecialEntity(Entity.*getEntityConfig*(*walkaround\_four*)));  
 }  
  
 @Override  
 **public void** updateMapEntities(MapManager mapMgr, Batch batch, **float** delta){  
 **for**(**int** i = 0; i < **mapEntities**.**size**; i++){  
 **mapEntities**.get(i).update(mapMgr, batch, delta);  
 }  
 }  
  
 @Override  
 **public void** unloadMusic() {  
  
 }  
  
 @Override  
 **public void** loadMusic() {  
  
 }  
  
}

**package** com.mygdx.game.maps;  
  
**import** com.badlogic.gdx.graphics.g2d.Batch;  
**import** com.mygdx.game.audio.AudioObserver;  
**import** com.mygdx.game.tools.Entity;  
  
**public class** IceForestMap **extends** Map {  
 **private static** String *mapPath* = **"tilemaps/ice\_forest.tmx"**;  
  
 **public** IceForestMap(){  
 **super**(MapFactory.MapType.***ICE\_FOREST***, *mapPath*);  
 }  
  
 @Override  
 **public void** updateMapEntities(MapManager mapMgr, Batch batch, **float** delta){  
 **for**( Entity entity : **mapEntities**){  
 entity.update(mapMgr, batch, delta);  
 }  
 }  
  
 @Override  
 **public void** unloadMusic() {  
 notify(AudioObserver.AudioCommand.***MUSIC\_STOP***, AudioObserver.AudioTypeEvent.***MUSIC\_ICE\_FOREST***);  
 }  
  
 @Override  
 **public void** loadMusic() {  
 notify(AudioObserver.AudioCommand.***MUSIC\_LOAD***, AudioObserver.AudioTypeEvent.***MUSIC\_ICE\_FOREST***);  
 notify(AudioObserver.AudioCommand.***MUSIC\_PLAY\_LOOP***, AudioObserver.AudioTypeEvent.***MUSIC\_ICE\_FOREST***);  
 }  
  
}

**package** com.mygdx.game.maps;  
  
**import** com.badlogic.gdx.graphics.g2d.Batch;  
**import** com.mygdx.game.tools.Entity;  
  
**public class** InnFirstFloorMap **extends** Map {  
 **private static** String *mapPath* = **"tilemaps/inn\_first\_floor.tmx"**;  
 **private static** String *herbalist* = **"json\_scripts/chara/herbalist.json"**;  
  
 InnFirstFloorMap(){  
 **super**(MapFactory.MapType.***INN\_FIRST\_FLOOR***, *mapPath*);  
 **mapEntities**.add(initSpecialEntity(Entity.*getEntityConfig*(*herbalist*)));  
 }  
  
 @Override  
 **public void** updateMapEntities(MapManager mapMgr, Batch batch, **float** delta){  
 **for**( Entity entity : **mapEntities**){  
 entity.update(mapMgr, batch, delta);  
 }  
 }  
  
 @Override  
 **public void** unloadMusic() {  
  
 }  
  
 @Override  
 **public void** loadMusic() {  
  
 }  
  
}

**package** com.mygdx.game.maps;  
  
**import** com.badlogic.gdx.graphics.g2d.Batch;  
**import** com.mygdx.game.tools.Entity;  
  
**public class** InnSecondFloorMap **extends** Map {  
 **private static** String *mapPath* = **"tilemaps/inn\_second\_floor.tmx"**;  
 **private static** String *inn\_clark* = **"json\_scripts/chara/inn\_clark.json"**;  
  
 InnSecondFloorMap(){  
 **super**(MapFactory.MapType.***INN\_SECOND\_FLOOR***, *mapPath*);  
 **mapEntities**.add(initSpecialEntity(Entity.*getEntityConfig*(*inn\_clark*)));  
 }  
  
 @Override  
 **public void** updateMapEntities(MapManager mapMgr, Batch batch, **float** delta){  
 **for**(**int** i = 0; i < **mapEntities**.**size**; i++){  
 **mapEntities**.get(i).update(mapMgr, batch, delta);  
 }  
 }  
  
 @Override  
 **public void** unloadMusic() {  
  
 }  
  
 @Override  
 **public void** loadMusic() {  
  
 }  
  
}

**package** com.mygdx.game.maps;  
  
**import** com.badlogic.gdx.graphics.g2d.Batch;  
**import** com.mygdx.game.audio.AudioObserver;  
**import** com.mygdx.game.tools.Entity;  
  
**public class** LavaMap **extends** Map {  
 **private static** String *mapPath* = **"tilemaps/lava.tmx"**;  
  
 **public** LavaMap(){  
 **super**(MapFactory.MapType.***LAVA***, *mapPath*);  
 }  
  
 @Override  
 **public void** updateMapEntities(MapManager mapMgr, Batch batch, **float** delta){  
 **for**( Entity entity : **mapEntities**){  
 entity.update(mapMgr, batch, delta);  
 }  
 }  
  
 @Override  
 **public void** unloadMusic() {  
 notify(AudioObserver.AudioCommand.***MUSIC\_STOP***, AudioObserver.AudioTypeEvent.***MUSIC\_LAVA***);  
 }  
  
 @Override  
 **public void** loadMusic() {  
 notify(AudioObserver.AudioCommand.***MUSIC\_LOAD***, AudioObserver.AudioTypeEvent.***MUSIC\_LAVA***);  
 notify(AudioObserver.AudioCommand.***MUSIC\_PLAY\_LOOP***, AudioObserver.AudioTypeEvent.***MUSIC\_LAVA***);  
 }  
  
}

**package** com.mygdx.game.maps;  
  
**import** com.badlogic.gdx.graphics.g2d.Batch;  
**import** com.mygdx.game.tools.Entity;  
  
**public class** LibraryMap **extends** Map{  
 **private static** String *mapPath* = **"tilemaps/library.tmx"**;  
 **private static** String *librarian* = **"json\_scripts/chara/librarian.json"**;  
  
 LibraryMap(){  
 **super**(MapFactory.MapType.***LIBRARY***, *mapPath*);  
 **mapEntities**.add(initSpecialEntity(Entity.*getEntityConfig*(*librarian*)));  
 }  
  
 @Override  
 **public void** updateMapEntities(MapManager mapMgr, Batch batch, **float** delta){  
 **for**( Entity entity : **mapEntities**){  
 entity.update(mapMgr, batch, delta);  
 }  
 }  
  
 @Override  
 **public void** unloadMusic() {  
  
 }  
  
 @Override  
 **public void** loadMusic() {  
  
 }  
}

**package** com.mygdx.game.maps;  
  
**import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.graphics.g2d.Batch;  
**import** com.badlogic.gdx.maps.MapLayer;  
**import** com.badlogic.gdx.maps.MapObject;  
**import** com.badlogic.gdx.maps.objects.RectangleMapObject;  
**import** com.badlogic.gdx.maps.tiled.TiledMap;  
**import** com.badlogic.gdx.math.Vector2;  
**import** com.badlogic.gdx.utils.Array;  
**import** com.badlogic.gdx.utils.Json;  
**import** com.mygdx.game.audio.AudioManager;  
**import** com.mygdx.game.audio.AudioObserver;  
**import** com.mygdx.game.audio.AudioSubject;  
**import** com.mygdx.game.components.Component;  
**import** com.mygdx.game.tools.Entity;  
**import** com.mygdx.game.tools.EntityConfig;  
**import** com.mygdx.game.tools.EntityFactory;  
**import** com.mygdx.game.tools.Utility;  
  
**import** java.util.Hashtable;  
  
**public abstract class** Map **implements** AudioSubject{  
 **private static final** String ***TAG*** = Map.**class**.getSimpleName();  
  
 **public final static float *UNIT\_SCALE*** = 1/16f;  
 **public static** String *specificPortal*;  
 **private** Array<AudioObserver> **observers**;  
  
 *//Map layers* **protected final static** String ***COLLISION\_LAYER*** = **"MAP\_COLLISION\_LAYER"**;  
 **protected final static** String ***SPAWNS\_LAYER*** = **"MAP\_SPAWNS\_LAYER"**;  
 **protected final static** String ***PORTAL\_LAYER*** = **"MAP\_PORTAL\_LAYER"**;  
 **protected final static** String ***ENEMY\_SPAWN\_LAYER*** = **"MAP\_ENEMY\_SPAWN\_LAYER"**;  
  
 *//Starting locations* **protected final static** String ***PLAYER\_START*** = **"PLAYER\_START"**;  
 **protected final static** String ***NPC\_START*** = **"NPC\_START"**;  
  
 **protected** Json **json**;  
  
 **protected** Vector2 **playerStartPositionRect**;  
 **protected** Vector2 **closestPlayerStartPosition**;  
 **protected** Vector2 **convertedUnits**;  
 **protected** TiledMap **currentMap** = **null**;  
 **protected** Vector2 **playerStart**;  
 **protected** Array<Vector2> **npcStartPositions**;  
 **protected** Hashtable<String, Vector2> **specialNPCStartPositions**;  
  
 **protected** MapLayer **collisionLayer** = **null**;  
 **protected** MapLayer **portalLayer** = **null**;  
 **protected** MapLayer **spawnsLayer** = **null**;  
 **protected** MapLayer **enemySpawnLayer** = **null**;  
  
 **protected** MapFactory.MapType **currentMapType**;  
 **protected** Array<Entity> **mapEntities**;  
  
 **public** Map(MapFactory.MapType mapType, String fullMapPath){  
 **json** = **new** Json();  
 **observers** = **new** Array<AudioObserver>();  
 **mapEntities** = **new** Array<Entity>(10);  
 **currentMapType** = mapType;  
 **playerStart** = **new** Vector2(0,0);  
 **playerStartPositionRect** = **new** Vector2(0,0);  
 **closestPlayerStartPosition** = **new** Vector2(0,0);  
 **convertedUnits** = **new** Vector2(0,0);  
  
 **if**( fullMapPath == **null** || fullMapPath.isEmpty() ) {  
 Gdx.*app*.debug(***TAG***, **"Map is invalid"**);  
 **return**;  
 }  
  
 Utility.*loadMapAsset*(fullMapPath);  
 **if**( Utility.*isAssetLoaded*(fullMapPath) ) {  
 **currentMap** = Utility.*getMapAsset*(fullMapPath);  
 }**else**{  
 Gdx.*app*.debug(***TAG***, **"Map not loaded"**);  
 **return**;  
 }  
  
 **collisionLayer** = **currentMap**.getLayers().get(***COLLISION\_LAYER***);  
 **if**( **collisionLayer** == **null** ){  
 Gdx.*app*.debug(***TAG***, **"No collision layer!"**);  
 }  
  
 **portalLayer** = **currentMap**.getLayers().get(***PORTAL\_LAYER***);  
 **if**( **portalLayer** == **null** ){  
 Gdx.*app*.debug(***TAG***, **"No portal layer!"**);  
 }  
  
 **spawnsLayer** = **currentMap**.getLayers().get(***SPAWNS\_LAYER***);  
 **if**( **spawnsLayer** == **null** ){  
 Gdx.*app*.debug(***TAG***, **"No spawn layer!"**);  
 }**else**{  
 setClosestStartPosition(**playerStart**);  
 }  
  
 **enemySpawnLayer** = **currentMap**.getLayers().get(***ENEMY\_SPAWN\_LAYER***);  
 **if**( **enemySpawnLayer** == **null** ){  
 Gdx.*app*.debug(***TAG***, **"No enemy layer found!"**);  
 }  
  
 **npcStartPositions** = getNPCStartPositions();  
 **specialNPCStartPositions** = getSpecialNPCStartPositions();  
  
 *//Observers* **this**.addObserver(AudioManager.*getInstance*());  
 }  
  
 **public abstract void** updateMapEntities(MapManager mapMgr, Batch batch, **float** delta);  
  
 **public** Entity initSpecialEntity(EntityConfig entityConfig){  
 Vector2 position = **new** Vector2(0,0);  
  
 **if**( **specialNPCStartPositions**.containsKey(entityConfig.getEntityID()) ) {  
 position = **specialNPCStartPositions**.get(entityConfig.getEntityID());  
 }  
 **return** initEntity(entityConfig, position);  
 }  
  
 **public** Entity initEntity(EntityConfig entityConfig, Vector2 position){  
 Entity entity = EntityFactory.*getEntity*(EntityFactory.EntityType.***NPC***);  
 entity.setEntityConfig(entityConfig);  
  
 entity.sendMessage(Component.MESSAGE.***LOAD\_ANIMATIONS***, **json**.toJson(entity.getEntityConfig()));  
 entity.sendMessage(Component.MESSAGE.***INIT\_START\_POSITION***, **json**.toJson(position));  
 entity.sendMessage(Component.MESSAGE.***INIT\_STATE***, **json**.toJson(entity.getEntityConfig().getState()));  
 entity.sendMessage(Component.MESSAGE.***INIT\_DIRECTION***, **json**.toJson(entity.getEntityConfig().getDirection()));  
  
 **return** entity;  
 }  
  
 **private** Array<Vector2> getNPCStartPositions(){  
 Array<Vector2> npcStartPositions = **new** Array<Vector2>();  
  
 **for**( MapObject object: **spawnsLayer**.getObjects()){  
 String objectName = object.getName();  
  
 **if**( objectName == **null** || objectName.isEmpty() ){  
 **continue**;  
 }  
  
 **if**( objectName.equalsIgnoreCase(***NPC\_START***) ){  
 *//Get center of rectangle* **float** x = ((RectangleMapObject)object).getRectangle().getX();  
 **float** y = ((RectangleMapObject)object).getRectangle().getY();  
  
 *//scale by the unit to convert from map coordinates* x \*= ***UNIT\_SCALE***;  
 y \*= ***UNIT\_SCALE***;  
  
 npcStartPositions.add(**new** Vector2(x,y));  
 }  
 }  
 **return** npcStartPositions;  
 }  
  
 **private** Hashtable<String, Vector2> getSpecialNPCStartPositions(){  
 Hashtable<String, Vector2> specialNPCStartPositions = **new** Hashtable<String, Vector2>();  
  
 **for**( MapObject object: **spawnsLayer**.getObjects()){  
 String objectName = object.getName();  
  
 **if**( objectName == **null** || objectName.isEmpty() ){  
 **continue**;  
 }  
  
 *//This is meant for all the special spawn locations, a catch all, so ignore known ones* **if**( objectName.equalsIgnoreCase(***NPC\_START***) ||  
 objectName.equalsIgnoreCase(***PLAYER\_START***) ){  
 **continue**;  
 }  
  
 *//Get center of rectangle* **float** x = ((RectangleMapObject)object).getRectangle().getX();  
 **float** y = ((RectangleMapObject)object).getRectangle().getY();  
  
 *//scale by the unit to convert from map coordinates* x \*= ***UNIT\_SCALE***;  
 y \*= ***UNIT\_SCALE***;  
  
 specialNPCStartPositions.put(objectName, **new** Vector2(x,y));  
 }  
 **return** specialNPCStartPositions;  
 }  
  
 *//****TODO speak about this* public void** setClosestStartPosition(**final** Vector2 position) {  
 Gdx.*app*.debug(***TAG***, **"setClosestStartPosition INPUT: ("** + position.**x** + **","** + position.**y** + **") "** + **currentMapType**.toString());  
  
 *//Get last known position on this map* **playerStartPositionRect**.set(0, 0);  
 **closestPlayerStartPosition**.set(0, 0);  
 **float** shortestDistance = 0f;  
  
 *//Go through all player start positions and choose closest to last known position* **for** (MapObject object : **spawnsLayer**.getObjects()) {  
 String objectName = object.getName();  
  
 **if** (objectName == **null** || objectName.isEmpty()) {  
 **continue**;  
 }  
  
 ((RectangleMapObject) object).getRectangle().getPosition(**playerStartPositionRect**);  
  
 **if** (objectName.equalsIgnoreCase(***PLAYER\_START***)) {  
 **closestPlayerStartPosition**.set(**playerStartPositionRect**);  
 **playerStart** = **closestPlayerStartPosition**.cpy();  
 }  
 **else if** (*specificPortal* != **null**) {  
 *//Gdx.app.debug(TAG, "specificPortal is not null");* **if** (*specificPortal*.equalsIgnoreCase(objectName)) {  
 Gdx.*app*.debug(***TAG***, **"specificPortal "** + *specificPortal* + **" activated"**);  
 **closestPlayerStartPosition**.set(**playerStartPositionRect**);  
 **playerStart** = **closestPlayerStartPosition**.cpy();  
 **return**;  
 }  
 }  
 }  
  
  
 }  
  
 **public void** setSpecificPortal(String specificPortal){  
 *//Gdx.app.debug(TAG, "portalProperties are not null");* **this**.*specificPortal* = specificPortal;  
 }  
  
 **public** Array<Entity> getMapEntities(){  
 **return mapEntities**;  
 }  
  
 **public** Vector2 getPlayerStart() {  
 **return playerStart**;  
 }  
  
 **public** Vector2 getPlayerStartUnitScaled(){  
 Vector2 playerStart = **this**.**playerStart**.cpy();  
 playerStart.set(**this**.**playerStart**.**x** \* ***UNIT\_SCALE***, **this**.**playerStart**.**y** \* ***UNIT\_SCALE***);  
 **return** playerStart;  
 }  
  
 **public void** setPlayerStart(Vector2 playerStart) {  
 **this**.**playerStart** = playerStart;  
 }  
  
 **public** MapLayer getCollisionLayer(){  
 **return collisionLayer**;  
 }  
  
 **public** MapLayer getEnemySpawnLayer() {  
 **return enemySpawnLayer**;  
 }  
  
 **public** MapLayer getPortalLayer(){  
 **return portalLayer**;  
 }  
  
 **public** TiledMap getCurrentTiledMap() {  
 **return currentMap**;  
 }  
  
 **abstract public void** unloadMusic();  
 **abstract public void** loadMusic();  
  
 @Override  
 **public void** addObserver(AudioObserver audioObserver) {  
 **observers**.add(audioObserver);  
 }  
  
 @Override  
 **public void** removeObserver(AudioObserver audioObserver) {  
 **observers**.removeValue(audioObserver, **true**);  
 }  
  
 @Override  
 **public void** removeAllObservers() {  
 **observers**.removeAll(**observers**, **true**);  
 }  
  
 @Override  
 **public void** notify(AudioObserver.AudioCommand command, AudioObserver.AudioTypeEvent event) {  
 **for**(AudioObserver observer: **observers**){  
 observer.onNotify(command, event);  
 }  
 }  
  
}

**package** com.mygdx.game.maps;  
  
**import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.maps.MapProperties;  
  
**import** java.util.Hashtable;  
  
**public class** MapFactory {  
  
 **private static final** String ***TAG*** = MapFactory.**class**.getSimpleName();  
  
 *//All maps for the game* **private static** Hashtable<MapType,Map> *mapTable* = **new** Hashtable<MapType, Map>();  
  
 **public static enum** MapType{  
 ***TOWN***,  
 ***ARMORER***,  
 ***LIBRARY***,  
 ***HERB\_SHOP***,  
 ***HOUSE\_ONE***,  
 ***INN\_FIRST\_FLOOR***,  
 ***INN\_SECOND\_FLOOR***,  
 ***TOP\_WORLD***,  
 ***FOREST***,  
 ***CAVE***,  
 ***COAST***,  
 ***ICE\_FOREST***,  
 ***DESERT\_TEMPLE***,  
 ***LAVA*** }  
  
 **public static** Map getMap(MapType mapType){  
 Map map = **null**;  
 **switch**(mapType){  
 **case *TOWN***:  
 *//Gdx.app.log(TAG, "mapType is: " + mapType.toString());* map = *mapTable*.get(MapType.***TOWN***);  
 **if**( map == **null** ){  
 map = **new** TownMap();  
 *mapTable*.put(MapType.***TOWN***, map);  
 }  
 **break**;  
 **case *ARMORER***:  
 map = *mapTable*.get(MapType.***ARMORER***);  
 **if**( map == **null** ){  
 map = **new** ArmorerMap();  
 *mapTable*.put(MapType.***ARMORER***, map);  
 }  
 **break**;  
 **case *LIBRARY***:  
 map = *mapTable*.get(MapType.***LIBRARY***);  
 **if**( map == **null** ){  
 map = **new** LibraryMap();  
 *mapTable*.put(MapType.***LIBRARY***, map);  
 }  
 **break**;  
 **case *HERB\_SHOP***:  
 map = *mapTable*.get(MapType.***HERB\_SHOP***);  
 **if**( map == **null** ){  
 map = **new** HerbShopMap();  
 *mapTable*.put(MapType.***HERB\_SHOP***, map);  
 }  
 **break**;  
 **case *HOUSE\_ONE***:  
 map = *mapTable*.get(MapType.***HOUSE\_ONE***);  
 **if**( map == **null** ){  
 map = **new** HouseOneMap();  
 *mapTable*.put(MapType.***HOUSE\_ONE***, map);  
 }  
 **break**;  
 **case *INN\_FIRST\_FLOOR***:  
 map = *mapTable*.get(MapType.***INN\_FIRST\_FLOOR***);  
 **if**( map == **null** ){  
 map = **new** InnFirstFloorMap();  
 *mapTable*.put(MapType.***INN\_FIRST\_FLOOR***, map);  
 }  
 **break**;  
 **case *INN\_SECOND\_FLOOR***:  
 map = *mapTable*.get(MapType.***INN\_SECOND\_FLOOR***);  
 **if**( map == **null** ){  
 map = **new** InnSecondFloorMap();  
 *mapTable*.put(MapType.***INN\_SECOND\_FLOOR***, map);  
 }  
 **break**;  
 **case *TOP\_WORLD***:  
 map = *mapTable*.get(MapType.***TOP\_WORLD***);  
 **if**( map == **null** ){  
 map = **new** TopworldMap();  
 *mapTable*.put(MapType.***TOP\_WORLD***, map);  
 }  
 **break**;  
 **case *FOREST***:  
 map = *mapTable*.get(MapType.***FOREST***);  
 **if**( map == **null** ){  
 map = **new** ForestMap();  
 *mapTable*.put(MapType.***FOREST***, map);  
 }  
 **break**;  
 **case *CAVE***:  
 map = *mapTable*.get(MapType.***CAVE***);  
 **if**( map == **null** ){  
 map = **new** CaveMap();  
 *mapTable*.put(MapType.***CAVE***, map);  
 }  
 **break**;  
 **case *COAST***:  
 map = *mapTable*.get(MapType.***COAST***);  
 **if**( map == **null** ){  
 map = **new** CoastMap();  
 *mapTable*.put(MapType.***COAST***, map);  
 }  
 **break**;  
 **case *ICE\_FOREST***:  
 map = *mapTable*.get(MapType.***ICE\_FOREST***);  
 **if**( map == **null** ){  
 map = **new** IceForestMap();  
 *mapTable*.put(MapType.***ICE\_FOREST***, map);  
 }  
 **break**;  
 **case *DESERT\_TEMPLE***:  
 map = *mapTable*.get(MapType.***DESERT\_TEMPLE***);  
 **if**( map == **null** ){  
 map = **new** DesertTempleMap();  
 *mapTable*.put(MapType.***DESERT\_TEMPLE***, map);  
 }  
 **break**;  
 **case *LAVA***:  
 map = *mapTable*.get(MapType.***LAVA***);  
 **if**( map == **null** ){  
 map = **new** LavaMap();  
 *mapTable*.put(MapType.***LAVA***, map);  
 }  
 **break**;  
 **default**:  
 **break**;  
  
 }  
 **return** map;  
 }  
  
 **public static void** clearCache(){  
 *mapTable*.clear();  
 }  
  
  
}

**package** com.mygdx.game.maps;  
  
**import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.graphics.Camera;  
**import** com.badlogic.gdx.graphics.g2d.Batch;  
**import** com.badlogic.gdx.maps.MapLayer;  
**import** com.badlogic.gdx.maps.MapProperties;  
**import** com.badlogic.gdx.maps.tiled.TiledMap;  
**import** com.badlogic.gdx.math.Vector2;  
**import** com.badlogic.gdx.utils.Array;  
**import** com.mygdx.game.tools.Entity;  
**import** com.mygdx.game.profile.ProfileManager;  
**import** com.mygdx.game.profile.ProfileObserver;  
  
**public class** MapManager **implements** ProfileObserver {  
  
 **private static final** String ***TAG*** = MapManager.**class**.getSimpleName();  
 **private boolean mapChanged** = **false**;  
 **private** Camera **camera**;  
 **private** Map **currentMap**;  
 **private** Entity **player**;  
  
 **public** MapManager(){  
 ProfileManager.*getInstance*().addObserver(**this**);  
 }  
  
 @Override  
 **public void** onNotify(ProfileManager profileManager, ProfileEvent event) {  
 **switch**(event){  
 **case *PROFILE\_LOADED***:  
 String currentMap = profileManager.getProperty(**"currentMapType"**, String.**class**);  
 MapFactory.MapType mapType;  
 **if**( currentMap == **null** || currentMap.isEmpty() ){  
 mapType = MapFactory.MapType.***TOWN***;  
 }**else**{  
 mapType = MapFactory.MapType.*valueOf*(currentMap);  
 }  
 loadMap(mapType);  
  
 Vector2 townMapStartPosition = profileManager.getProperty(**"townMapStartPosition"**, Vector2.**class**);  
 **if**( townMapStartPosition != **null** ){  
 MapFactory.*getMap*(MapFactory.MapType.***TOWN***).setPlayerStart(townMapStartPosition);  
 }  
  
 Vector2 armorerMapStartPosition = profileManager.getProperty(**"armorerMapStartPosition"**, Vector2.**class**);  
 **if**( armorerMapStartPosition != **null** ){  
 MapFactory.*getMap*(MapFactory.MapType.***ARMORER***).setPlayerStart(armorerMapStartPosition);  
 }  
  
 Vector2 libraryMapStartPosition = profileManager.getProperty(**"libraryMapStartPosition"**, Vector2.**class**);  
 **if**( libraryMapStartPosition != **null** ){  
 MapFactory.*getMap*(MapFactory.MapType.***LIBRARY***).setPlayerStart(libraryMapStartPosition);  
 }  
  
 Vector2 herbShopMapStartPosition = profileManager.getProperty(**"herbShopMapStartPosition"**, Vector2.**class**);  
 **if**( herbShopMapStartPosition != **null** ){  
 MapFactory.*getMap*(MapFactory.MapType.***HERB\_SHOP***).setPlayerStart(herbShopMapStartPosition);  
 }  
  
 Vector2 houseOneStartPosition = profileManager.getProperty(**"houseOneStartPosition"**, Vector2.**class**);  
 **if**( houseOneStartPosition != **null** ){  
 MapFactory.*getMap*(MapFactory.MapType.***HOUSE\_ONE***).setPlayerStart(houseOneStartPosition);  
 }  
  
 Vector2 innFirstFloorMapStartPosition = profileManager.getProperty(**"innFirstFloorMapStartPosition"**, Vector2.**class**);  
 **if**( innFirstFloorMapStartPosition != **null** ){  
 MapFactory.*getMap*(MapFactory.MapType.***INN\_FIRST\_FLOOR***).setPlayerStart(innFirstFloorMapStartPosition);  
 }  
  
 Vector2 innSecondFloorMapStartPosition = profileManager.getProperty(**"innSecondFloorMapStartPosition"**, Vector2.**class**);  
 **if**( innFirstFloorMapStartPosition != **null** ){  
 MapFactory.*getMap*(MapFactory.MapType.***INN\_SECOND\_FLOOR***).setPlayerStart(innSecondFloorMapStartPosition);  
 }  
  
 Vector2 topWorldMapStartPosition = profileManager.getProperty(**"topWorldMapStartPosition"**, Vector2.**class**);  
 **if**( topWorldMapStartPosition != **null** ){  
 MapFactory.*getMap*(MapFactory.MapType.***TOP\_WORLD***).setPlayerStart(topWorldMapStartPosition);  
 }  
  
 Vector2 forestMapStartPosition = profileManager.getProperty(**"forestMapStartPosition"**, Vector2.**class**);  
 **if**( forestMapStartPosition != **null** ){  
 MapFactory.*getMap*(MapFactory.MapType.***FOREST***).setPlayerStart(forestMapStartPosition);  
 }  
 Vector2 caveMapStartPosition = profileManager.getProperty(**"caveMapStartPosition"**, Vector2.**class**);  
 **if**( caveMapStartPosition != **null** ){  
 MapFactory.*getMap*(MapFactory.MapType.***CAVE***).setPlayerStart(caveMapStartPosition);  
 }  
 Vector2 coastMapStartPosition = profileManager.getProperty(**"coastMapStartPosition"**, Vector2.**class**);  
 **if**( coastMapStartPosition != **null** ){  
 MapFactory.*getMap*(MapFactory.MapType.***COAST***).setPlayerStart(coastMapStartPosition);  
 }  
 Vector2 iceForestMapStartPosition = profileManager.getProperty(**"iceForestMapStartPosition"**, Vector2.**class**);  
 **if**( iceForestMapStartPosition != **null** ){  
 MapFactory.*getMap*(MapFactory.MapType.***ICE\_FOREST***).setPlayerStart(iceForestMapStartPosition);  
 }  
 Vector2 desertTempleMapStartPosition = profileManager.getProperty(**"desertTempleMapStartPosition"**, Vector2.**class**);  
 **if**( desertTempleMapStartPosition != **null** ){  
 MapFactory.*getMap*(MapFactory.MapType.***DESERT\_TEMPLE***).setPlayerStart(desertTempleMapStartPosition);  
 }  
 Vector2 lavaMapStartPosition = profileManager.getProperty(**"lavaMapStartPosition"**, Vector2.**class**);  
 **if**( lavaMapStartPosition != **null** ){  
 MapFactory.*getMap*(MapFactory.MapType.***LAVA***).setPlayerStart(lavaMapStartPosition);  
 }  
  
 **break**;  
 **case *SAVING\_PROFILE***:  
 **if**( **this**.**currentMap** != **null** ){  
 profileManager.setProperty(**"currentMapType"**, **this**.**currentMap**.**currentMapType**.toString());  
 }  
  
 profileManager.setProperty(**"townMapStartPosition"**, MapFactory.*getMap*(MapFactory.MapType.***TOWN***).getPlayerStart() );  
 profileManager.setProperty(**"armorerMapStartPosition"**, MapFactory.*getMap*(MapFactory.MapType.***ARMORER***).getPlayerStart() );  
 profileManager.setProperty(**"libraryMapStartPosition"**, MapFactory.*getMap*(MapFactory.MapType.***LIBRARY***).getPlayerStart() );  
 profileManager.setProperty(**"herbShopMapStartPosition"**, MapFactory.*getMap*(MapFactory.MapType.***HERB\_SHOP***).getPlayerStart() );  
 profileManager.setProperty(**"houseOneStartPosition"**, MapFactory.*getMap*(MapFactory.MapType.***HOUSE\_ONE***).getPlayerStart() );  
 profileManager.setProperty(**"innFirstFloorMapStartPosition"**, MapFactory.*getMap*(MapFactory.MapType.***INN\_FIRST\_FLOOR***).getPlayerStart() );  
 profileManager.setProperty(**"innSecondFloorMapStartPosition"**, MapFactory.*getMap*(MapFactory.MapType.***INN\_SECOND\_FLOOR***).getPlayerStart() );  
 profileManager.setProperty(**"topWorldMapStartPosition"**, MapFactory.*getMap*(MapFactory.MapType.***TOP\_WORLD***).getPlayerStart() );  
 profileManager.setProperty(**"forestMapStartPosition"**, MapFactory.*getMap*(MapFactory.MapType.***FOREST***).getPlayerStart() );  
 profileManager.setProperty(**"caveMapStartPosition"**, MapFactory.*getMap*(MapFactory.MapType.***CAVE***).getPlayerStart() );  
 profileManager.setProperty(**"coastMapStartPosition"**, MapFactory.*getMap*(MapFactory.MapType.***COAST***).getPlayerStart() );  
 profileManager.setProperty(**"iceForestMapStartPosition"**, MapFactory.*getMap*(MapFactory.MapType.***ICE\_FOREST***).getPlayerStart() );  
 profileManager.setProperty(**"desertTempleMapStartPosition"**, MapFactory.*getMap*(MapFactory.MapType.***DESERT\_TEMPLE***).getPlayerStart() );  
 profileManager.setProperty(**"lavaMapStartPosition"**, MapFactory.*getMap*(MapFactory.MapType.***LAVA***).getPlayerStart() );  
  
  
  
 **break**;  
 **case *CLEAR\_CURRENT\_PROFILE***:  
 **this**.**currentMap** = **null**;  
 profileManager.setProperty(**"currentMapType"**, MapFactory.MapType.***TOWN***.toString());  
  
 MapFactory.*clearCache*();  
  
 profileManager.setProperty(**"townMapStartPosition"**, MapFactory.*getMap*(MapFactory.MapType.***TOWN***).getPlayerStart() );  
 profileManager.setProperty(**"armorerMapStartPosition"**, MapFactory.*getMap*(MapFactory.MapType.***ARMORER***).getPlayerStart() );  
 profileManager.setProperty(**"libraryMapStartPosition"**, MapFactory.*getMap*(MapFactory.MapType.***LIBRARY***).getPlayerStart() );  
 profileManager.setProperty(**"herbShopMapStartPosition"**, MapFactory.*getMap*(MapFactory.MapType.***HERB\_SHOP***).getPlayerStart() );  
 profileManager.setProperty(**"houseOneStartPosition"**, MapFactory.*getMap*(MapFactory.MapType.***HOUSE\_ONE***).getPlayerStart() );  
 profileManager.setProperty(**"innFirstFloorMapStartPosition"**, MapFactory.*getMap*(MapFactory.MapType.***INN\_FIRST\_FLOOR***).getPlayerStart() );  
 profileManager.setProperty(**"innSecondFloorMapStartPosition"**, MapFactory.*getMap*(MapFactory.MapType.***INN\_SECOND\_FLOOR***).getPlayerStart() );  
 profileManager.setProperty(**"topWorldMapStartPosition"**, MapFactory.*getMap*(MapFactory.MapType.***TOP\_WORLD***).getPlayerStart() );  
 profileManager.setProperty(**"forestMapStartPosition"**, MapFactory.*getMap*(MapFactory.MapType.***FOREST***).getPlayerStart() );  
 profileManager.setProperty(**"caveMapStartPosition"**, MapFactory.*getMap*(MapFactory.MapType.***CAVE***).getPlayerStart() );  
 profileManager.setProperty(**"coastMapStartPosition"**, MapFactory.*getMap*(MapFactory.MapType.***COAST***).getPlayerStart() );  
 profileManager.setProperty(**"iceForestMapStartPosition"**, MapFactory.*getMap*(MapFactory.MapType.***ICE\_FOREST***).getPlayerStart() );  
 profileManager.setProperty(**"desertTempleMapStartPosition"**, MapFactory.*getMap*(MapFactory.MapType.***DESERT\_TEMPLE***).getPlayerStart() );  
 profileManager.setProperty(**"lavaMapStartPosition"**, MapFactory.*getMap*(MapFactory.MapType.***LAVA***).getPlayerStart() );  
  
 **break**;  
 **default**:  
 **break**;  
 }  
 }  
  
 **public** TiledMap getCurrentTiledMap(){  
 **if**( **currentMap** == **null** ) {  
 loadMap(MapFactory.MapType.***TOWN***);  
 }  
 **return currentMap**.getCurrentTiledMap();  
 }  
  
 **public** MapLayer getCollisionLayer(){  
 **return currentMap**.getCollisionLayer();  
 }  
  
 **public** MapLayer getPortalLayer(){  
 **return currentMap**.getPortalLayer();  
 }  
  
 **public void** loadMap(MapFactory.MapType mapType){  
 Map map = MapFactory.*getMap*(mapType);  
 map.setClosestStartPosition(map.**playerStart**);  
  
 **if**( map == **null** ){  
 Gdx.*app*.debug(***TAG***, **"Map does not exist! "**);  
 **return**;  
 }  
  
 **if**( **currentMap** != **null** ){  
 **currentMap**.unloadMusic();  
 }  
  
 map.loadMusic();  
  
  
 **currentMap** = map;  
 **mapChanged** = **true**;  
 Gdx.*app*.debug(***TAG***, **"Player Start: ("** + **currentMap**.getPlayerStart().**x** + **","** + **currentMap**.getPlayerStart().**y** + **")"**);  
 }  
  
 **public void** disableCurrentmapMusic(){  
 **currentMap**.unloadMusic();  
 }  
  
 **public void** enableCurrentmapMusic(){  
 **currentMap**.loadMusic();  
 }  
  
 **public** Vector2 getPlayerStartUnitScaled() {  
 **return currentMap**.getPlayerStartUnitScaled();  
 }  
  
 **public void** updateCurrentMapEntities(MapManager mapMgr, Batch batch, **float** delta){  
 **currentMap**.updateMapEntities(mapMgr, batch, delta);  
 }  
  
 **public final** Array<Entity> getCurrentMapEntities(){  
 **return currentMap**.getMapEntities();  
 }  
  
 **public** MapLayer getEnemySpawnLayer(){  
 **return currentMap**.getEnemySpawnLayer();  
 }  
  
 **public void** setPlayer(Entity entity){  
 **this**.**player** = entity;  
 }  
  
 **public** Entity getPlayer(){  
 **return this**.**player**;  
 }  
  
 **public void** setCamera(Camera camera){  
 **this**.**camera** = camera;  
 }  
  
 **public** Camera getCamera(){  
 **return camera**;  
 }  
  
 **public boolean** hasMapChanged(){  
 **return mapChanged**;  
 }  
  
 **public void** setMapChanged(**boolean** hasMapChanged){  
 **this**.**mapChanged** = hasMapChanged;  
 }  
  
 **public void** setSpecificPortal(String specificPortal) {  
 *//Gdx.app.debug(TAG, "portalProperties are not null");* **currentMap**.setSpecificPortal(specificPortal);  
 }  
  
}

**package** com.mygdx.game.maps;  
  
**import** com.badlogic.gdx.graphics.g2d.Batch;  
**import** com.mygdx.game.audio.AudioObserver;  
**import** com.mygdx.game.tools.Entity;  
  
**public class** TopworldMap **extends** Map {  
 **private static** String *mapPath* = **"tilemaps/topworld.tmx"**;  
  
 **public** TopworldMap(){  
 **super**(MapFactory.MapType.***TOP\_WORLD***, *mapPath*);  
 }  
  
 @Override  
 **public void** updateMapEntities(MapManager mapMgr, Batch batch, **float** delta){  
 **for**( Entity entity : **mapEntities**){  
 entity.update(mapMgr, batch, delta);  
 }  
 }  
  
  
 @Override  
 **public void** unloadMusic() {  
 notify(AudioObserver.AudioCommand.***MUSIC\_STOP***, AudioObserver.AudioTypeEvent.***MUSIC\_TOPWORLD***);  
 }  
  
 @Override  
 **public void** loadMusic() {  
 notify(AudioObserver.AudioCommand.***MUSIC\_LOAD***, AudioObserver.AudioTypeEvent.***MUSIC\_TOPWORLD***);  
 notify(AudioObserver.AudioCommand.***MUSIC\_PLAY\_LOOP***, AudioObserver.AudioTypeEvent.***MUSIC\_TOPWORLD***);  
 }  
}

**package** com.mygdx.game.maps;  
  
**import** com.badlogic.gdx.graphics.g2d.Batch;  
**import** com.badlogic.gdx.math.Vector2;  
**import** com.mygdx.game.audio.AudioObserver;  
**import** com.mygdx.game.components.PlayerPhysicsComponent;  
**import** com.mygdx.game.tools.Entity;  
  
**public class** TownMap **extends** Map {  
 **private static final** String ***TAG*** = PlayerPhysicsComponent.**class**.getSimpleName();  
  
 **private static** String *mapPath* = **"tilemaps/town.tmx"**;  
 **private static** String *walkaround\_one* = **"json\_scripts/chara/walkaround\_one.json"**;  
 **private static** String *walkaround\_two* = **"json\_scripts/chara/walkaround\_two.json"**;  
 **private static** String *walkaround\_three* = **"json\_scripts/chara/walkaround\_three.json"**;  
 **private static** String *walkaround\_four* = **"json\_scripts/chara/walkaround\_four.json"**;  
 **private static** String *inn\_clark* = **"json\_scripts/chara/inn\_clark.json"**;  
  
 **public** TownMap(){  
 **super**(MapFactory.MapType.***TOWN***, *mapPath*);  
  
 **for**( Vector2 position: **npcStartPositions**){  
 **mapEntities**.add(initEntity(Entity.*getEntityConfig*(*walkaround\_one*), position));  
 }  
  
 *//Special cases* **mapEntities**.add(initSpecialEntity(Entity.*getEntityConfig*(*walkaround\_two*)));  
 **mapEntities**.add(initSpecialEntity(Entity.*getEntityConfig*(*walkaround\_three*)));  
 **mapEntities**.add(initSpecialEntity(Entity.*getEntityConfig*(*walkaround\_four*)));  
 **mapEntities**.add(initSpecialEntity(Entity.*getEntityConfig*(*inn\_clark*)));  
 }  
  
 @Override  
 **public void** updateMapEntities(MapManager mapMgr, Batch batch, **float** delta){  
 **for**(**int** i = 0; i < **mapEntities**.**size**; i++){  
 **mapEntities**.get(i).update(mapMgr, batch, delta);  
 }  
 }  
  
 @Override  
 **public void** unloadMusic() {  
 notify(AudioObserver.AudioCommand.***MUSIC\_STOP***, AudioObserver.AudioTypeEvent.***MUSIC\_TOWN***);  
 }  
  
 @Override  
 **public void** loadMusic() {  
 notify(AudioObserver.AudioCommand.***MUSIC\_LOAD***, AudioObserver.AudioTypeEvent.***MUSIC\_TOWN***);  
 notify(AudioObserver.AudioCommand.***MUSIC\_PLAY\_LOOP***, AudioObserver.AudioTypeEvent.***MUSIC\_TOWN***);  
 }  
  
}

## src.com.mygdx.game.profile

**package** com.mygdx.game.profile;  
  
**import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.files.FileHandle;  
**import** com.badlogic.gdx.utils.Array;  
**import** com.badlogic.gdx.utils.Json;  
**import** com.badlogic.gdx.utils.ObjectMap;  
  
**import** java.util.Enumeration;  
**import** java.util.Hashtable;  
  
**public class** ProfileManager **extends** ProfileSubject {  
 **private** Json **json**;  
 **private static** ProfileManager *profileManager*;  
 **private** Hashtable<String,FileHandle> **profiles** = **null**;  
 **private** ObjectMap<String, Object> **profileProperties** = **new** ObjectMap<String, Object>();  
 **private** String **profileName**;  
 **private boolean isNewProfile** = **false**;  
  
 **private static final** String ***SAVEGAME\_SUFFIX*** = **".sav"**;  
 **public static final** String ***DEFAULT\_PROFILE*** = **"default"**;  
  
  
 **private** ProfileManager(){  
 **json** = **new** Json();  
 **profiles** = **new** Hashtable<String,FileHandle>();  
 **profiles**.clear();  
 **profileName** = ***DEFAULT\_PROFILE***;  
 storeAllProfiles();  
 }  
  
 **public static final** ProfileManager getInstance(){  
 **if**( *profileManager* == **null**){  
 *profileManager* = **new** ProfileManager();  
 }  
 **return** *profileManager*;  
 }  
  
 **public void** setIsNewProfile(**boolean** isNewProfile){  
 **this**.**isNewProfile** = isNewProfile;  
 }  
  
 **public boolean** getIsNewProfile(){  
 **return this**.**isNewProfile**;  
 }  
  
 **public** Array<String> getProfileList(){  
 Array<String> profiles = **new** Array<String>();  
 **for** (Enumeration<String> e = **this**.**profiles**.keys(); e.hasMoreElements();){  
 profiles.add(e.nextElement());  
 }  
 **return** profiles;  
 }  
  
 **public** FileHandle getProfileFile(String profile){  
 **if**( !doesProfileExist(profile) ){  
 **return null**;  
 }  
 **return profiles**.get(profile);  
 }  
  
 **public void** storeAllProfiles(){  
 **if**( Gdx.*files*.isLocalStorageAvailable() ){  
 FileHandle[] files = Gdx.*files*.local(**"."**).list(***SAVEGAME\_SUFFIX***);  
  
 **for**(FileHandle file: files) {  
 **profiles**.put(file.nameWithoutExtension(), file);  
 }  
 }**else**{  
 *//****TODO: try external directory here* return**;  
 }  
 }  
  
 **public boolean** doesProfileExist(String profileName){  
 **return profiles**.containsKey(profileName);  
 }  
  
 **public void** writeProfileToStorage(String profileName, String fileData, **boolean** overwrite){  
 String fullFilename = profileName+***SAVEGAME\_SUFFIX***;  
  
 **boolean** localFileExists = Gdx.*files*.local(fullFilename).exists();  
  
 *//If we cannot overwrite and the file exists, exit* **if**( localFileExists && !overwrite ){  
 **return**;  
 }  
  
 FileHandle file = **null**;  
  
 **if**( Gdx.*files*.isLocalStorageAvailable() ) {  
 file = Gdx.*files*.local(fullFilename);  
 file.writeString(fileData, !overwrite);  
 }  
  
 **profiles**.put(profileName, file);  
 }  
  
 **public void** setProperty(String key, Object object){  
 **profileProperties**.put(key, object);  
 }  
  
 **public** <T **extends** Object> T getProperty(String key, Class<T> type){  
 T property = **null**;  
 **if**( !**profileProperties**.containsKey(key) ){  
 **return** property;  
 }  
 property = (T) **profileProperties**.get(key);  
 **return** property;  
 }  
  
 **public void** saveProfile(){  
 notify(**this**, ProfileObserver.ProfileEvent.***SAVING\_PROFILE***);  
 String text = **json**.prettyPrint(**json**.toJson(**profileProperties**));  
 writeProfileToStorage(**profileName**, text, **true**);  
 }  
  
 **public void** loadProfile(){  
 **if**(**isNewProfile**){  
 notify(**this**, ProfileObserver.ProfileEvent.***CLEAR\_CURRENT\_PROFILE***);  
 saveProfile();  
 }  
  
 String fullProfileFileName = **profileName** +***SAVEGAME\_SUFFIX***;  
 **boolean** doesProfileFileExist = Gdx.*files*.local(fullProfileFileName).exists();  
  
 **if**( !doesProfileFileExist ){  
 System.***out***.println(**"File doesn't exist!"**);  
 **return**;  
 }  
  
 **profileProperties** = **json**.fromJson(ObjectMap.**class**, **profiles**.get(**profileName**));  
 notify(**this**, ProfileObserver.ProfileEvent.***PROFILE\_LOADED***);  
 **isNewProfile** = **false**;  
 }  
  
 **public void** setCurrentProfile(String profileName){  
 **if**( doesProfileExist(profileName) ){  
 **this**.**profileName** = profileName;  
 }**else**{  
 **this**.**profileName** = ***DEFAULT\_PROFILE***;  
 }  
 }  
  
}

**package** com.mygdx.game.profile;  
  
**public interface** ProfileObserver {  
 **public static enum** ProfileEvent{  
 ***PROFILE\_LOADED***,  
 ***SAVING\_PROFILE***,  
 ***CLEAR\_CURRENT\_PROFILE*** }  
  
 **void** onNotify(**final** ProfileManager profileManager, ProfileEvent event);  
}

**package** com.mygdx.game.profile;  
  
**import** com.badlogic.gdx.utils.Array;  
  
**public class** ProfileSubject {  
  
 **private** Array<ProfileObserver> **observers**;  
  
 **public** ProfileSubject(){  
 **observers** = **new** Array<ProfileObserver>();  
 }  
  
 **public void** addObserver(ProfileObserver profileObserver){  
 **observers**.add(profileObserver);  
 }  
  
 **public void** removeObserver(ProfileObserver profileObserver){  
 **observers**.removeValue(profileObserver, **true**);  
 }  
  
 **protected void** notify(**final** ProfileManager profileManager, ProfileObserver.ProfileEvent event){  
 **for**(ProfileObserver observer: **observers**){  
 observer.onNotify(profileManager, event);  
 }  
 }  
  
}

## src.com.mygdx.game.screens

**package** com.mygdx.game.screens;  
  
**import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.Screen;  
**import** com.badlogic.gdx.graphics.GL20;  
**import** com.badlogic.gdx.scenes.scene2d.InputEvent;  
**import** com.badlogic.gdx.scenes.scene2d.InputListener;  
**import** com.badlogic.gdx.scenes.scene2d.Stage;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Label;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Table;  
**import** com.badlogic.gdx.scenes.scene2d.ui.TextButton;  
**import** com.badlogic.gdx.utils.Align;  
**import** com.mygdx.game.FinalKanjiQuest;  
**import** com.mygdx.game.audio.AudioObserver;  
**import** com.mygdx.game.tools.Utility;  
  
**public class** GameOverScreen **extends** GameScreen {  
 **private** Stage **stage**;  
 **private final** FinalKanjiQuest **game**;  
 *//private static final String DEATH\_MESSAGE = "You have fought bravely, but alas, you have fallen during your epic struggle.";* **private static final** String ***GAMEOVER*** = **"Game Over"**;  
  
 **public** GameOverScreen(FinalKanjiQuest fkq){  
 **this**.**game** = fkq;  
  
 *//create* **stage** = **new** Stage();  
 TextButton continueButton = **new** TextButton(**"Continue"**, Utility.*GUI\_SKINS*);  
 TextButton mainMenuButton = **new** TextButton(**"Main Menu"**, Utility.*GUI\_SKINS*);  
 *//Label messageLabel = new Label(DEATH\_MESSAGE, Utility.GUI\_SKINS);  
 //messageLabel.setWrap(true);* Label gameOverLabel = **new** Label(***GAMEOVER***, Utility.*GUI\_SKINS*);  
 gameOverLabel.setAlignment(Align.***center***);  
  
 Table table = **new** Table();  
  
 *//Layout* table.setFillParent(**true**);  
 *//table.add(messageLabel).pad(50, 50,50,50).expandX().fillX().row();* table.add(gameOverLabel);  
 table.row();  
 table.add(continueButton).pad(50,50,10,50);  
 table.row();  
 table.add(mainMenuButton).pad(10,50,50,50);  
  
 **stage**.addActor(table);  
  
 *//Listeners* continueButton.addListener(**new** InputListener() {  
 @Override  
 **public boolean** touchDown(InputEvent event, **float** x, **float** y, **int** pointer, **int** button) {  
 **game**.setScreen(**game**.getScreenType(FinalKanjiQuest.ScreenType.***LoadGame***));  
 **return true**;  
 }  
 }  
 );  
  
 mainMenuButton.addListener(**new** InputListener() {  
  
 @Override  
 **public boolean** touchDown(InputEvent event, **float** x, **float** y, **int** pointer, **int** button) {  
 **game**.setScreen(**game**.getScreenType(FinalKanjiQuest.ScreenType.***MainMenu***));  
 **return true**;  
 }  
 }  
 );  
  
 notify(AudioObserver.AudioCommand.***MUSIC\_LOAD***, AudioObserver.AudioTypeEvent.***MUSIC\_TITLE***);  
  
 }  
  
 @Override  
 **public void** render(**float** delta) {  
 **if**( delta == 0){  
 **return**;  
 }  
  
 Gdx.*gl*.glClearColor(0, 0, 0, 1);  
 Gdx.*gl*.glClear(GL20.***GL\_COLOR\_BUFFER\_BIT***);  
 **stage**.act(delta);  
 **stage**.draw();  
 }  
  
 @Override  
 **public void** resize(**int** width, **int** height) {  
 **stage**.getViewport().setScreenSize(width, height);  
 }  
  
 @Override  
 **public void** show() {  
 Gdx.*input*.setInputProcessor(**stage**);  
 notify(AudioObserver.AudioCommand.***MUSIC\_PLAY\_LOOP***, AudioObserver.AudioTypeEvent.***MUSIC\_TITLE***);  
 }  
  
 @Override  
 **public void** hide() {  
 Gdx.*input*.setInputProcessor(**null**);  
 }  
  
 @Override  
 **public void** pause() {  
 }  
  
 @Override  
 **public void** resume() {  
 }  
  
 @Override  
 **public void** dispose() {  
 **stage**.clear();  
 **stage**.dispose();  
 }  
}

**package** com.mygdx.game.screens;  
  
**import** com.badlogic.gdx.Screen;  
**import** com.badlogic.gdx.utils.Array;  
**import** com.mygdx.game.audio.AudioManager;  
**import** com.mygdx.game.audio.AudioObserver;  
**import** com.mygdx.game.audio.AudioSubject;  
  
**public class** GameScreen **implements** Screen, AudioSubject {  
 **private** Array<AudioObserver> **observers**;  
  
 **public** GameScreen(){  
 **observers** = **new** Array<AudioObserver>();  
 **this**.addObserver(AudioManager.*getInstance*());  
 }  
  
 @Override  
 **public void** addObserver(AudioObserver audioObserver) {  
 **observers**.add(audioObserver);  
 }  
  
 @Override  
 **public void** removeObserver(AudioObserver audioObserver) {  
 **observers**.removeValue(audioObserver, **true**);  
 }  
  
 @Override  
 **public void** removeAllObservers() {  
 **observers**.removeAll(**observers**, **true**);  
 }  
  
 @Override  
 **public void** notify(AudioObserver.AudioCommand command, AudioObserver.AudioTypeEvent event) {  
 **for**(AudioObserver observer: **observers**){  
 observer.onNotify(command, event);  
 }  
 }  
  
 @Override  
 **public void** show() {  
  
 }  
  
 @Override  
 **public void** render(**float** delta) {  
  
 }  
  
 @Override  
 **public void** resize(**int** width, **int** height) {  
  
 }  
  
 @Override  
 **public void** pause() {  
  
 }  
  
 @Override  
 **public void** resume() {  
  
 }  
  
 @Override  
 **public void** hide() {  
  
 }  
  
 @Override  
 **public void** dispose() {  
  
 }  
}

**package** com.mygdx.game.screens;  
  
**import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.Screen;  
**import** com.badlogic.gdx.files.FileHandle;  
**import** com.badlogic.gdx.graphics.GL20;  
**import** com.badlogic.gdx.graphics.Texture;  
**import** com.badlogic.gdx.graphics.g2d.Sprite;  
**import** com.badlogic.gdx.scenes.scene2d.InputEvent;  
**import** com.badlogic.gdx.scenes.scene2d.InputListener;  
**import** com.badlogic.gdx.scenes.scene2d.Stage;  
**import** com.badlogic.gdx.scenes.scene2d.ui.List;  
**import** com.badlogic.gdx.scenes.scene2d.ui.ScrollPane;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Table;  
**import** com.badlogic.gdx.scenes.scene2d.ui.TextButton;  
**import** com.badlogic.gdx.utils.Array;  
**import** com.mygdx.game.FinalKanjiQuest;  
**import** com.mygdx.game.audio.AudioObserver;  
**import** com.mygdx.game.tools.Utility;  
**import** com.mygdx.game.profile.ProfileManager;  
  
  
**public class** LoadGameScreen **extends** GameScreen {  
 **private** Stage **stage**;  
 **private final** FinalKanjiQuest **game**;  
  
 **private** Texture **texture**;  
 **private** Sprite **backgroundSprite**;  
 **private final** List **listItems**;  
  
 **public** LoadGameScreen(FinalKanjiQuest fkq){  
 **this**.**game** = fkq;  
  
 **texture** = **new** Texture(Gdx.*files*.internal(**"sprites/maps/topworld.png"**));  
 **backgroundSprite** = **new** Sprite(**texture**);  
  
 *//create* **stage** = **new** Stage();  
 TextButton loadButton = **new** TextButton(**"Load"**, Utility.*GUI\_SKINS*);  
 TextButton backButton = **new** TextButton(**"Back"**, Utility.*GUI\_SKINS*);  
  
 ProfileManager.*getInstance*().storeAllProfiles();  
 Array<String> list = ProfileManager.*getInstance*().getProfileList();  
 **listItems** = **new** List(Utility.*GUI\_SKINS*);  
 **listItems**.setItems(list);  
 ScrollPane scrollPane = **new** ScrollPane(**listItems**);  
  
 scrollPane.setOverscroll(**false**, **false**);  
 scrollPane.setFadeScrollBars(**false**);  
 scrollPane.setScrollingDisabled(**true**, **false**);  
 scrollPane.setScrollbarsOnTop(**true**);  
  
 Table table = **new** Table();  
 Table bottomTable = **new** Table();  
  
 *//Layout* table.center();  
 table.setFillParent(**true**);  
 table.padBottom(loadButton.getHeight());  
 table.add(scrollPane).center();  
  
 bottomTable.setHeight(loadButton.getHeight());  
 bottomTable.setWidth(Gdx.*graphics*.getWidth());  
 bottomTable.center();  
 bottomTable.add(loadButton).padRight(50);  
 bottomTable.add(backButton);  
  
 **stage**.addActor(table);  
 **stage**.addActor(bottomTable);  
  
 *//Listeners* backButton.addListener(**new** InputListener() {  
 @Override  
 **public boolean** touchDown(InputEvent event, **float** x, **float** y, **int** pointer, **int** button) {  
 **game**.setScreen(**game**.getScreenType(FinalKanjiQuest.ScreenType.***MainMenu***));  
 **return true**;  
 }  
 }  
 );  
  
 loadButton.addListener(**new** InputListener() {  
 @Override  
 **public boolean** touchDown(InputEvent event, **float** x, **float** y, **int** pointer, **int** button) {  
 **if**(**listItems**.getSelected() != **null**) {  
 String fileName = **listItems**.getSelected().toString();  
 **if** (fileName != **null**) {  
 FileHandle file = ProfileManager.*getInstance*().getProfileFile(fileName);  
 **if** (file != **null**) {  
 ProfileManager.*getInstance*().setCurrentProfile(fileName);  
 LoadGameScreen.**this**.notify(AudioObserver.AudioCommand.***MUSIC\_STOP***, AudioObserver.AudioTypeEvent.***MUSIC\_TITLE***);  
 **game**.setScreen(**game**.getScreenType(FinalKanjiQuest.ScreenType.***MainGame***));  
 }  
 }  
 }  
 **return true**;  
 }  
 }  
 );  
 }  
   
 @Override  
 **public void** render(**float** delta) {  
 **if**( delta == 0){  
 **return**;  
 }  
   
 Gdx.gl.glClearColor(0, 0, 0, 1);  
 Gdx.gl.glClear(GL20.GL\_COLOR\_BUFFER\_BIT);  
  
 stage.getBatch().begin();  
 stage.getBatch().draw(backgroundSprite, 0, 0, stage.getWidth(), stage.getHeight());  
 stage.getBatch().end();  
  
  
 stage.act(delta);  
 stage.draw();  
 }  
  
 @Override  
 **public void** resize(**int** width, **int** height) {  
 stage.getViewport().setScreenSize(width, height);  
 }  
  
 @Override  
 **public void** show() {  
 Array<String> list = ProfileManager.getInstance().getProfileList();  
 listItems.setItems(list);  
 Gdx.input.setInputProcessor(stage);  
 }  
  
 @Override  
 **public void** hide() {  
 Gdx.input.setInputProcessor(**null**);  
 }  
  
 @Override  
 **public void** pause() {  
 }  
  
 @Override  
 **public void** resume() {  
 }  
  
 @Override  
 **public void** dispose() {  
 stage.clear();  
 stage.dispose();  
 }  
  
}

**package** com.mygdx.game.screens;  
  
**import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.InputMultiplexer;  
**import** com.badlogic.gdx.Screen;  
**import** com.badlogic.gdx.graphics.GL20;  
**import** com.badlogic.gdx.graphics.OrthographicCamera;  
**import** com.badlogic.gdx.maps.tiled.renderers.OrthogonalTiledMapRenderer;  
**import** com.badlogic.gdx.utils.Json;  
**import** com.mygdx.game.audio.AudioManager;  
**import** com.mygdx.game.audio.AudioObserver;  
**import** com.mygdx.game.components.Component;  
**import** com.mygdx.game.tools.Entity;  
**import** com.mygdx.game.tools.EntityFactory;  
**import** com.mygdx.game.FinalKanjiQuest;  
**import** com.mygdx.game.gui.PlayerHUD;  
**import** com.mygdx.game.maps.Map;  
**import** com.mygdx.game.maps.MapManager;  
**import** com.mygdx.game.profile.ProfileManager;  
  
**public class** MainGameScreen **extends** GameScreen {  
  
 **private static final** String ***TAG*** = MainGameScreen.**class**.getSimpleName();  
  
 **private static class** VIEWPORT {  
 **static float** *viewportWidth*;  
 **static float** *viewportHeight*;  
 **static float** *virtualWidth*;  
 **static float** *virtualHeight*;  
 **static float** *physicalWidth*;  
 **static float** *physicalHeight*;  
 **static float** *aspectRatio*;  
 }  
  
 **public static enum** GameState {  
 ***SAVING***,  
 ***LOADING***,  
 ***RUNNING***,  
 ***PAUSED***,  
 ***GAME\_OVER***,  
  
 }  
 **private static** GameState *gameState*;  
  
 **protected** OrthogonalTiledMapRenderer **mapRenderer** = **null**;  
 **protected** OrthographicCamera **camera** = **null**;  
 **protected** OrthographicCamera **hudCamera** = **null**;  
 **protected** MapManager **mapMgr**;  
  
  
 **private** Json **json**;  
 **private final** FinalKanjiQuest **game**;  
 **private** InputMultiplexer **multiplexer**;  
  
 **private** Entity **player**;  
 **private** PlayerHUD **playerHUD**;  
  
 **public** MainGameScreen(FinalKanjiQuest fkq){  
 **this**.**game** = fkq;  
 **mapMgr** = **new** MapManager();  
 **json** = **new** Json();  
  
 *gameState* = GameState.***RUNNING***;  
 *//\_camera setup* setupViewport(15, 15);  
  
 *//get the current size* **camera** = **new** OrthographicCamera();  
 **camera**.setToOrtho(**false**, VIEWPORT.*viewportWidth*, VIEWPORT.*viewportHeight*);  
  
 **player** = EntityFactory.*getEntity*(EntityFactory.EntityType.***PLAYER***);  
 **mapMgr**.setPlayer(**player**);  
 **mapMgr**.setCamera(**camera**);  
  
 **hudCamera** = **new** OrthographicCamera();  
 **hudCamera**.setToOrtho(**false**, VIEWPORT.*physicalWidth*, VIEWPORT.*physicalHeight*);  
  
  
 **multiplexer** = **new** InputMultiplexer();  
 **playerHUD** = **new** PlayerHUD(**hudCamera**, **player**, **multiplexer**, **mapMgr**);  
 }  
  
 @Override  
 **public void** show() {  
 ProfileManager.*getInstance*().addObserver(**mapMgr**);  
 ProfileManager.*getInstance*().addObserver(**playerHUD**);  
  
 *setGameState*(GameState.***LOADING***);  
 Gdx.*input*.setInputProcessor(**multiplexer**);  
  
 **if**( **mapRenderer** == **null** ){  
 **mapRenderer** = **new** OrthogonalTiledMapRenderer(**mapMgr**.getCurrentTiledMap(), Map.***UNIT\_SCALE***);  
 }  
 }  
  
 @Override  
 **public void** hide() {  
 **if**( *gameState* != GameState.***GAME\_OVER*** ){  
 *setGameState*(GameState.***SAVING***);  
 }  
  
 Gdx.*input*.setInputProcessor(**null**);  
 }  
  
 @Override  
 **public void** render(**float** delta) {  
 **if**( *gameState* == GameState.***GAME\_OVER*** ){  
 **game**.setScreen(**game**.getScreenType(FinalKanjiQuest.ScreenType.***GameOver***));  
 }  
  
 **if**( *gameState* == GameState.***PAUSED*** ){  
 **player**.updateInput(delta);  
 **playerHUD**.render(delta);  
 *//mapMgr.disableCurrentmapMusic();* **return**;  
 }  
  
 Gdx.*gl*.glClearColor(0, 0, 0, 1);  
 Gdx.*gl*.glClear(GL20.***GL\_COLOR\_BUFFER\_BIT***);  
  
 **mapRenderer**.setView(**camera**);  
  
 **if**( **mapMgr**.hasMapChanged() ){  
 **mapRenderer**.setMap(**mapMgr**.getCurrentTiledMap());  
 **player**.sendMessage(Component.MESSAGE.***INIT\_START\_POSITION***, **json**.toJson(**mapMgr**.getPlayerStartUnitScaled()));  
  
 **camera**.**position**.set(**mapMgr**.getPlayerStartUnitScaled().**x**, **mapMgr**.getPlayerStartUnitScaled().**y**, 0f);  
 **camera**.update();  
  
 **mapMgr**.setMapChanged(**false**);  
 }  
  
 **mapRenderer**.render();  
  
 **mapMgr**.updateCurrentMapEntities(**mapMgr**, **mapRenderer**.getBatch(), delta );  
  
 **player**.update(**mapMgr**, **mapRenderer**.getBatch(), delta);  
 **playerHUD**.render(delta);  
 }  
  
 @Override  
 **public void** resize(**int** width, **int** height) {  
 setupViewport(15, 15);  
 **camera**.setToOrtho(**false**, VIEWPORT.*viewportWidth*, VIEWPORT.*viewportHeight*);  
 **playerHUD**.resize((**int**) VIEWPORT.*physicalWidth*, (**int**) VIEWPORT.*physicalHeight*);  
 }  
  
 @Override  
 **public void** pause() {  
 *setGameState*(GameState.***SAVING***);  
 }  
  
 @Override  
 **public void** resume() {  
 *setGameState*(GameState.***LOADING***);  
 }  
  
 @Override  
 **public void** dispose() {  
 **if**( **player** != **null** ){  
 **player**.unregisterObservers();  
 **player**.dispose();  
 }  
  
 **if**( **mapRenderer** != **null** ){  
 **mapRenderer**.dispose();  
 }  
  
 AudioManager.*getInstance*().dispose();  
 }  
  
 **public static void** setGameState(GameState state){  
 **switch**(state){  
 **case *RUNNING***:  
 *gameState* = GameState.***RUNNING***;  
 **break**;  
 **case *LOADING***:  
 *gameState* = GameState.***RUNNING***;  
 ProfileManager.*getInstance*().loadProfile();  
 **break**;  
 **case *SAVING***:  
 ProfileManager.*getInstance*().saveProfile();  
 *gameState* = GameState.***PAUSED***;  
 **break**;  
 **case *PAUSED***:  
 **if**( *gameState* == GameState.***PAUSED*** ){  
 *gameState* = GameState.***RUNNING***;  
 }**else if**( *gameState* == GameState.***RUNNING*** ){  
 *gameState* = GameState.***PAUSED***;  
 }  
 **break**;  
 **case *GAME\_OVER***:  
 *gameState* = GameState.***GAME\_OVER***;  
 **break**;  
 **default**:  
 *gameState* = GameState.***RUNNING***;  
 **break**;  
 }  
  
 }  
  
 **private void** setupViewport(**int** width, **int** height){  
 *//Make the viewport a percentage of the total display area* VIEWPORT.*virtualWidth* = width;  
 VIEWPORT.*virtualHeight* = height;  
  
 *//Current viewport dimensions* VIEWPORT.*viewportWidth* = VIEWPORT.*virtualWidth*;  
 VIEWPORT.*viewportHeight* = VIEWPORT.*virtualHeight*;  
  
 *//pixel dimensions of display* VIEWPORT.*physicalWidth* = Gdx.*graphics*.getWidth();  
 VIEWPORT.*physicalHeight* = Gdx.*graphics*.getHeight();  
  
 *//aspect ratio for current viewport* VIEWPORT.*aspectRatio* = (VIEWPORT.*virtualWidth* / VIEWPORT.*virtualHeight*);  
  
 *//update viewport if there could be skewing* **if**( VIEWPORT.*physicalWidth* / VIEWPORT.*physicalHeight* >= VIEWPORT.*aspectRatio*){  
 *//Letterbox left and right* VIEWPORT.*viewportWidth* = VIEWPORT.*viewportHeight* \* (VIEWPORT.*physicalWidth*/ VIEWPORT.*physicalHeight*);  
 VIEWPORT.*viewportHeight* = VIEWPORT.*virtualHeight*;  
 }**else**{  
 *//letterbox above and below* VIEWPORT.*viewportWidth* = VIEWPORT.*virtualWidth*;  
 VIEWPORT.*viewportHeight* = VIEWPORT.*viewportWidth* \* (VIEWPORT.*physicalHeight*/ VIEWPORT.*physicalWidth*);  
 }  
  
 Gdx.*app*.debug(***TAG***, **"WorldRenderer: virtual: ("** + VIEWPORT.*virtualWidth* + **","** + VIEWPORT.*virtualHeight* + **")"** );  
 Gdx.*app*.debug(***TAG***, **"WorldRenderer: viewport: ("** + VIEWPORT.*viewportWidth* + **","** + VIEWPORT.*viewportHeight* + **")"** );  
 Gdx.*app*.debug(***TAG***, **"WorldRenderer: physical: ("** + VIEWPORT.*physicalWidth* + **","** + VIEWPORT.*physicalHeight* + **")"** );  
 }  
  
}

**package** com.mygdx.game.screens;  
  
**import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.Screen;  
**import** com.badlogic.gdx.graphics.GL20;  
**import** com.badlogic.gdx.graphics.Texture;  
**import** com.badlogic.gdx.graphics.g2d.Sprite;  
**import** com.badlogic.gdx.scenes.scene2d.InputEvent;  
**import** com.badlogic.gdx.scenes.scene2d.InputListener;  
**import** com.badlogic.gdx.scenes.scene2d.Stage;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Label;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Table;  
**import** com.badlogic.gdx.scenes.scene2d.ui.TextButton;  
**import** com.mygdx.game.FinalKanjiQuest;  
**import** com.mygdx.game.FinalKanjiQuest.ScreenType;  
**import** com.mygdx.game.audio.AudioObserver;  
**import** com.mygdx.game.tools.Utility;  
  
**public class** MainMenuScreen **extends** GameScreen {  
  
 **private** Stage **stage**;  
 **private final** FinalKanjiQuest **game**;  
  
 **private** Texture **texture**;  
 **private** Sprite **backgroundSprite**;  
  
 **public** MainMenuScreen(FinalKanjiQuest fkq){  
 **this**.**game** = fkq;  
  
 *//creation* **stage** = **new** Stage();  
 Table table = **new** Table();  
 table.setFillParent(**true**);  
  
 **texture** = **new** Texture(Gdx.*files*.internal(**"sprites/maps/topworld.png"**));  
 **backgroundSprite** = **new** Sprite(**texture**);  
  
  
 Label title = **new** Label(**"FKQ"**, Utility.*GUI\_SKINS*, **"title\_text"**);  
 TextButton newGameButton = **new** TextButton(**"New Game"**, Utility.*GUI\_SKINS*);  
 TextButton loadGameButton = **new** TextButton(**"Load Game"**, Utility.*GUI\_SKINS*);  
 TextButton exitButton = **new** TextButton(**"Exit"**, Utility.*GUI\_SKINS*);  
  
 *//Layout* table.add(title).spaceBottom(75).row();  
 table.add(newGameButton).spaceBottom(10).row();  
 table.add(loadGameButton).spaceBottom(10).row();  
 table.add(exitButton).spaceBottom(10).row();  
  
 **stage**.addActor(table);  
  
 *//Listeners* newGameButton.addListener(**new** InputListener() {  
 @Override  
 **public boolean** touchDown(InputEvent event, **float** x, **float** y, **int** pointer, **int** button) {  
 **game**.setScreen(**game**.getScreenType(ScreenType.***NewGame***));  
 **return true**;  
 }  
 }  
 );  
  
 loadGameButton.addListener(**new** InputListener() {  
  
 @Override  
 **public boolean** touchDown(InputEvent event, **float** x, **float** y, **int** pointer, **int** button) {  
 **game**.setScreen(**game**.getScreenType(ScreenType.***LoadGame***));  
 **return true**;  
 }  
 }  
 );  
  
 exitButton.addListener(**new** InputListener() {  
  
 @Override  
 **public boolean** touchDown(InputEvent event, **float** x, **float** y, **int** pointer, **int** button) {  
 Gdx.*app*.exit();  
 **return true**;  
 }  
  
 }  
 );  
  
  
 notify(AudioObserver.AudioCommand.***MUSIC\_LOAD***, AudioObserver.AudioTypeEvent.***MUSIC\_TITLE***);  
  
 }  
   
 @Override  
 **public void** render(**float** delta) {  
 Gdx.*gl*.glClearColor(0, 0, 0, 1);  
 Gdx.*gl*.glClear(GL20.***GL\_COLOR\_BUFFER\_BIT***);  
  
 **stage**.getBatch().begin();  
 **stage**.getBatch().draw(**backgroundSprite**, 0, 0, **stage**.getWidth(), **stage**.getHeight());  
 **stage**.getBatch().end();  
  
 **stage**.act(delta);  
 **stage**.draw();  
 }  
  
 @Override  
 **public void** resize(**int** width, **int** height) {  
 **stage**.getViewport().setScreenSize(width, height);  
 }  
  
 @Override  
 **public void** show() {  
 notify(AudioObserver.AudioCommand.***MUSIC\_PLAY\_LOOP***, AudioObserver.AudioTypeEvent.***MUSIC\_TITLE***);  
 Gdx.*input*.setInputProcessor(**stage**);  
 }  
  
 @Override  
 **public void** hide() {  
 Gdx.*input*.setInputProcessor(**null**);  
 }  
  
 @Override  
 **public void** pause() {  
 }  
  
 @Override  
 **public void** resume() {  
 }  
  
 @Override  
 **public void** dispose() {  
 **stage**.dispose();  
 }  
   
}

**package** com.mygdx.game.screens;  
  
**import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.Screen;  
**import** com.badlogic.gdx.graphics.GL20;  
**import** com.badlogic.gdx.graphics.Texture;  
**import** com.badlogic.gdx.graphics.g2d.Sprite;  
**import** com.badlogic.gdx.scenes.scene2d.InputEvent;  
**import** com.badlogic.gdx.scenes.scene2d.InputListener;  
**import** com.badlogic.gdx.scenes.scene2d.Stage;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Dialog;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Label;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Table;  
**import** com.badlogic.gdx.scenes.scene2d.ui.TextButton;  
**import** com.badlogic.gdx.scenes.scene2d.ui.TextField;  
**import** com.mygdx.game.FinalKanjiQuest;  
**import** com.mygdx.game.FinalKanjiQuest.ScreenType;  
**import** com.mygdx.game.audio.AudioObserver;  
**import** com.mygdx.game.tools.Utility;  
**import** com.mygdx.game.profile.ProfileManager;  
  
**public class** NewGameScreen **extends** GameScreen {  
  
 **private** Stage **stage**;  
 **private final** FinalKanjiQuest **game**;  
  
 **private** Texture **texture**;  
 **private** Sprite **backgroundSprite**;  
 **private final** TextField **profileText**;  
 **private final** Table **middleTable**;  
  
 **public** NewGameScreen(FinalKanjiQuest fkq){  
 **this**.**game** = fkq;  
 **stage** = **new** Stage();  
  
 *//TopTable* **texture** = **new** Texture(Gdx.*files*.internal(**"sprites/maps/topworld.png"**));  
 **backgroundSprite** = **new** Sprite(**texture**);  
  
 Label profileName = **new** Label(**"Enter Profile Name: "**, Utility.*GUI\_SKINS*);  
 **profileText** = **new** TextField(**""**, Utility.*GUI\_SKINS*);  
 **profileText**.setMaxLength(20);  
  
 Table topTable = **new** Table();  
 topTable.padBottom(**stage**.getHeight()/2);  
 topTable.setFillParent(**true**);  
 topTable.add(profileName).center();  
 topTable.row();  
 topTable.add(**profileText**).fill();  
  
 *//MiddleTable* Label overwriteLabel = **new** Label(**"Overwrite existing profile name?"**, Utility.*GUI\_SKINS*);  
 TextButton cancelButton = **new** TextButton(**"Cancel"**, Utility.*GUI\_SKINS*, **"inventory"**);  
 TextButton overwriteButton = **new** TextButton(**"Overwrite"**, Utility.*GUI\_SKINS*, **"inventory"**);  
  
 **middleTable** = **new** Dialog(**"Overwrite?"**, Utility.*GUI\_SKINS*);  
 **middleTable**.setWidth(Gdx.*graphics*.getWidth()/1.05f);  
 **middleTable**.setHeight(Gdx.*graphics*.getHeight()/4);  
 **middleTable**.setPosition(Gdx.*graphics*.getWidth()/40f, Gdx.*graphics*.getHeight()/4);  
  
 **middleTable**.row();  
 **middleTable**.add(overwriteLabel).left();  
 **middleTable**.row();  
 **middleTable**.add(overwriteButton).bottom().left();  
 **middleTable**.add(cancelButton).bottom().right();  
 **middleTable**.setVisible(**false**);  
  
 *//bottomTable* TextButton startButton = **new** TextButton(**"Start"**, Utility.*GUI\_SKINS*);  
 TextButton backButton = **new** TextButton(**"Back"**, Utility.*GUI\_SKINS*);  
  
 Table bottomTable = **new** Table();  
 bottomTable.setHeight(startButton.getHeight());  
 bottomTable.setWidth(Gdx.*graphics*.getWidth());  
 bottomTable.center();  
 bottomTable.add(startButton).padRight(50);  
 bottomTable.add(backButton);  
  
 **stage**.addActor(topTable);  
 **stage**.addActor(**middleTable**);  
 **stage**.addActor(bottomTable);  
  
 *//Listeners* cancelButton.addListener(**new** InputListener() {  
  
 @Override  
 **public boolean** touchDown(InputEvent event, **float** x, **float** y, **int** pointer, **int** button ){  
 **middleTable**.setVisible(**false**);  
 **return true**;  
 }  
 }  
 );  
  
 overwriteButton.addListener(**new** InputListener() {  
  
 @Override  
 **public boolean** touchDown(InputEvent event, **float** x, **float** y, **int** pointer, **int** button) {  
 String messageText = **profileText**.getText();  
 ProfileManager.*getInstance*().writeProfileToStorage(messageText, **""**, **true**);  
 ProfileManager.*getInstance*().setCurrentProfile(messageText);  
 ProfileManager.*getInstance*().setIsNewProfile(**true**);  
 **middleTable**.setVisible(**false**);  
 NewGameScreen.**this**.notify(AudioObserver.AudioCommand.***MUSIC\_STOP***, AudioObserver.AudioTypeEvent.***MUSIC\_TITLE***);  
 **game**.setScreen(**game**.getScreenType(ScreenType.***MainGame***));  
 **return true**;  
 }  
 }  
 );  
  
 startButton.addListener(**new** InputListener() {  
  
 @Override  
 **public boolean** touchDown(InputEvent event, **float** x, **float** y, **int** pointer, **int** button ){  
 String messageText = **profileText**.getText();  
  
 **if**(messageText.length() < 1){  
 **return true**;  
 }  
  
 *//check to see if the current profile matches one that already exists* **boolean** exists = **false**;  
  
 exists = ProfileManager.*getInstance*().doesProfileExist(messageText);  
  
 **if**( exists ){  
 *//Pop up dialog for Overwrite* **middleTable**.setVisible(**true**);  
 }**else**{  
 ProfileManager.*getInstance*().writeProfileToStorage(messageText,**""**,**false**);  
 ProfileManager.*getInstance*().setCurrentProfile(messageText);  
 ProfileManager.*getInstance*().setIsNewProfile(**true**);  
 NewGameScreen.**this**.notify(AudioObserver.AudioCommand.***MUSIC\_STOP***, AudioObserver.AudioTypeEvent.***MUSIC\_TITLE***);  
 **game**.setScreen(**game**.getScreenType(ScreenType.***MainGame***));  
 }  
  
 **return true**;  
 }  
 }  
 );  
  
 backButton.addListener(**new** InputListener() {  
  
 @Override  
 **public boolean** touchDown(InputEvent event, **float** x, **float** y, **int** pointer, **int** button) {  
 **game**.setScreen(**game**.getScreenType(ScreenType.***MainMenu***));  
 **return true**;  
 }  
 }  
 );  
  
 }  
  
 @Override  
 **public void** render(**float** delta) {  
 **if**( delta == 0){  
 **return**;  
 }  
 Gdx.*gl*.glClearColor(0, 0, 0, 1);  
 Gdx.*gl*.glClear(GL20.***GL\_COLOR\_BUFFER\_BIT***);  
  
 **stage**.getBatch().begin();  
 **stage**.getBatch().draw(**backgroundSprite**, 0, 0, **stage**.getWidth(), **stage**.getHeight());  
 **stage**.getBatch().end();  
  
  
 **stage**.act(delta);  
 **stage**.draw();  
 }  
  
 @Override  
 **public void** resize(**int** width, **int** height) {  
 **stage**.getViewport().setScreenSize(width, height);  
 }  
  
 @Override  
 **public void** show() {  
 **middleTable**.setVisible(**false**);  
 **profileText**.setText(**""**);  
 Gdx.*input*.setInputProcessor(**stage**);  
 }  
  
 @Override  
 **public void** hide() {  
 **middleTable**.setVisible(**false**);  
 **profileText**.setText(**""**);  
 Gdx.*input*.setInputProcessor(**null**);  
 }  
  
 @Override  
 **public void** pause() {  
 }  
  
 @Override  
 **public void** resume() {  
 }  
  
 @Override  
 **public void** dispose() {  
 **stage**.clear();  
 **stage**.dispose();  
 }  
  
  
  
}

## src.com.mygdx.game.tools

**package** com.mygdx.game.tools;  
  
**import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.InputProcessor;  
**import** com.badlogic.gdx.graphics.g2d.Animation;  
**import** com.badlogic.gdx.graphics.g2d.Batch;  
**import** com.badlogic.gdx.math.MathUtils;  
**import** com.badlogic.gdx.math.Rectangle;  
**import** com.badlogic.gdx.math.Vector2;  
**import** com.badlogic.gdx.utils.Array;  
**import** com.badlogic.gdx.utils.Json;  
**import** com.badlogic.gdx.utils.JsonValue;  
**import** com.mygdx.game.components.Component;  
**import** com.mygdx.game.components.ComponentObserver;  
**import** com.mygdx.game.components.GraphicsComponent;  
**import** com.mygdx.game.components.InputComponent;  
**import** com.mygdx.game.components.PhysicsComponent;  
**import** com.mygdx.game.maps.MapManager;  
**import** com.mygdx.game.profile.ProfileManager;  
  
**import** java.util.ArrayList;  
**import** java.util.Hashtable;  
  
**public class** Entity {  
 **private static final** String ***TAG*** = Entity.**class**.getSimpleName();  
  
 **public static enum** Direction {  
 ***UP***,  
 ***UP\_LEFT***,  
 ***UP\_RIGHT***,  
 ***RIGHT***,  
 ***DOWN***,  
 ***DOWN\_LEFT***,  
 ***DOWN\_RIGHT***,  
 ***LEFT***;  
  
 **static public** Direction getRandomNext() {  
 **return** Direction.*values*()[MathUtils.*random*(Direction.*values*().**length** - 1)];  
 }  
  
 **public** Direction getOpposite() {  
 **if**( **this** == ***LEFT***){  
 **return *RIGHT***;  
 }**else if**( **this** == ***RIGHT***){  
 **return *LEFT***;  
 }**else if**( **this** == ***UP***){  
 **return *DOWN***;  
 }**else if**( **this** == ***UP\_LEFT***){  
 **return *DOWN\_RIGHT***;  
 }  
 **else if**( **this** == ***UP\_RIGHT***){  
 **return *DOWN\_LEFT***;  
 }  
 **else if**( **this** == ***DOWN\_LEFT***){  
 **return *UP\_RIGHT***;  
 }  
 **else if**( **this** == ***DOWN\_RIGHT***){  
 **return *UP\_LEFT***;  
 }  
 **else** {  
 **return *UP***;  
 }  
 }  
 }  
  
 **public static enum** State {  
 ***IDLE***,  
 ***WALKING***,  
  
 ***IMMOBILE***;*//This should always be last* **static public** State getRandomNext() {  
 *//Ignore IMMOBILE which should be last state* **return** State.*values*()[MathUtils.*random*(State.*values*().**length** - 2)];  
 }  
 }  
  
  
 **public static enum** AnimationType {  
 ***WALK\_LEFT***,  
 ***WALK\_RIGHT***,  
 ***WALK\_UP***,  
 ***WALK\_UP\_LEFT***,  
 ***WALK\_UP\_RIGHT***,  
 ***WALK\_DOWN***,  
 ***WALK\_DOWN\_LEFT***,  
 ***WALK\_DOWN\_RIGHT***,  
 ***IDLE***,  
 ***IMMOBILE*** }  
  
 **public static int** *FRAME\_WIDTH* = 23;  
 **public static int** *FRAME\_HEIGHT* = 34;  
 **private static final int *MAX\_COMPONENTS*** = 5;  
  
 **private** Json **\_json**;  
 **private** EntityConfig **\_entityConfig**;  
 **private** Array<Component> **components**;  
 **private** InputComponent **inputComponent**;  
 **private** GraphicsComponent **graphicsComponent**;  
 **private** PhysicsComponent **physicsComponent**;  
  
 **public** Entity(Entity entity){  
 set(entity);  
 }  
  
 **private** Entity set(Entity entity) {  
 **inputComponent** = entity.**inputComponent**;  
 **graphicsComponent** = entity.**graphicsComponent**;  
 **physicsComponent** = entity.**physicsComponent**;  
 **components** = entity.**components**;  
 **\_json** = entity.**\_json**;  
  
 **\_entityConfig** = **new** EntityConfig(entity.**\_entityConfig**);  
 **return this**;  
 }  
  
 **public** Entity(InputComponent inputComponent, PhysicsComponent physicsComponent, GraphicsComponent graphicsComponent){  
 **\_entityConfig** = **new** EntityConfig();  
 **\_json** = **new** Json();  
  
 **components** = **new** Array<Component>(***MAX\_COMPONENTS***);  
  
 **this**.**inputComponent** = inputComponent;  
 **this**.**physicsComponent** = physicsComponent;  
 **this**.**graphicsComponent** = graphicsComponent;  
  
 **components**.add(**this**.**inputComponent**);  
 **components**.add(**this**.**physicsComponent**);  
 **components**.add(**this**.**graphicsComponent**);  
 }  
  
 **public** EntityConfig getEntityConfig() {  
 **return \_entityConfig**;  
 }  
  
 **public void** sendMessage(Component.MESSAGE messageType, String ... args){  
 String fullMessage = messageType.toString();  
  
 **for** (String string : args) {  
 fullMessage += Component.***MESSAGE\_TOKEN*** + string;  
 }  
  
 **for**(Component component: **components**){  
 component.receiveMessage(fullMessage);  
 }  
 }  
  
 **public void** registerObserver(ComponentObserver observer){  
 **inputComponent**.addObserver(observer);  
 **physicsComponent**.addObserver(observer);  
 **graphicsComponent**.addObserver(observer);  
 }  
  
 **public void** unregisterObservers(){  
 **inputComponent**.removeAllObservers();  
 **physicsComponent**.removeAllObservers();  
 **graphicsComponent**.removeAllObservers();  
 }  
  
 **public void** update(MapManager mapMgr, Batch batch, **float** delta){  
 **inputComponent**.update(**this**, delta);  
 **physicsComponent**.update(**this**, mapMgr, delta);  
 **graphicsComponent**.update(**this**, mapMgr, batch, delta);  
 }  
  
 **public void** updateInput(**float** delta){  
 **inputComponent**.update(**this**, delta);  
 }  
  
 **public void** dispose(){  
 **for**(Component component: **components**){  
 component.dispose();  
 }  
 }  
  
 **public** Rectangle getCurrentBoundingBox(){  
 **return physicsComponent**.**boundingBox**;  
 }  
  
 **public** Vector2 getCurrentPosition(){  
 **return graphicsComponent**.**currentPosition**;  
 }  
  
 **public** InputProcessor getInputProcessor(){  
 **return inputComponent**;  
 }  
  
 **public void** setEntityConfig(EntityConfig entityConfig){  
 **this**.**\_entityConfig** = entityConfig;  
 }  
  
 **public** Animation getAnimation(Entity.AnimationType type){  
 **return graphicsComponent**.getAnimation(type);  
 }  
  
 **static public** EntityConfig getEntityConfig(String configFilePath){  
 Json json = **new** Json();  
 **return** json.fromJson(EntityConfig.**class**, Gdx.*files*.internal(configFilePath));  
 }  
  
 **static public** Array<EntityConfig> getEntityConfigs(String configFilePath){  
 Json json = **new** Json();  
 Array<EntityConfig> configs = **new** Array<EntityConfig>();  
  
 ArrayList<JsonValue> list = json.fromJson(ArrayList.**class**, Gdx.*files*.internal(configFilePath));  
  
 **for** (JsonValue jsonVal : list) {  
 configs.add(json.readValue(EntityConfig.**class**, jsonVal));  
 }  
  
 **return** configs;  
 }  
  
 **public static** EntityConfig loadEntityConfigByPath(String entityConfigPath){  
 EntityConfig entityConfig = Entity.*getEntityConfig*(entityConfigPath);  
 EntityConfig serializedConfig = ProfileManager.*getInstance*().getProperty(entityConfig.getEntityID(), EntityConfig.**class**);  
  
 **if**( serializedConfig == **null** ){  
 **return** entityConfig;  
 }**else**{  
 **return** serializedConfig;  
 }  
 }  
  
 **public static** EntityConfig loadEntityConfig(EntityConfig entityConfig){  
 EntityConfig serializedConfig = ProfileManager.*getInstance*().getProperty(entityConfig.getEntityID(), EntityConfig.**class**);  
  
 **if**( serializedConfig == **null** ){  
 **return** entityConfig;  
 }**else**{  
 **return** serializedConfig;  
 }  
 }  
  
 **public static** Entity initEntity(EntityConfig entityConfig, Vector2 position){  
 Json json = **new** Json();  
 Entity entity = EntityFactory.*getEntity*(EntityFactory.EntityType.***NPC***);  
 entity.setEntityConfig(entityConfig);  
  
 entity.sendMessage(Component.MESSAGE.***LOAD\_ANIMATIONS***, json.toJson(entity.getEntityConfig()));  
 entity.sendMessage(Component.MESSAGE.***INIT\_START\_POSITION***, json.toJson(position));  
 entity.sendMessage(Component.MESSAGE.***INIT\_STATE***, json.toJson(entity.getEntityConfig().getState()));  
 entity.sendMessage(Component.MESSAGE.***INIT\_DIRECTION***, json.toJson(entity.getEntityConfig().getDirection()));  
  
 **return** entity;  
 }  
  
 **public static** Hashtable<String, Entity> initEntities(Array<EntityConfig> configs){  
 Json json = **new** Json();  
 Hashtable<String, Entity > entities = **new** Hashtable<String, Entity>();  
 **for**( EntityConfig config: configs ){  
 Entity entity = EntityFactory.*getEntity*(EntityFactory.EntityType.***NPC***);  
  
 entity.setEntityConfig(config);  
 entity.sendMessage(Component.MESSAGE.***LOAD\_ANIMATIONS***, json.toJson(entity.getEntityConfig()));  
 entity.sendMessage(Component.MESSAGE.***INIT\_START\_POSITION***, json.toJson(**new** Vector2(0,0)));  
 entity.sendMessage(Component.MESSAGE.***INIT\_STATE***, json.toJson(entity.getEntityConfig().getState()));  
 entity.sendMessage(Component.MESSAGE.***INIT\_DIRECTION***, json.toJson(entity.getEntityConfig().getDirection()));  
  
 entities.put(entity.getEntityConfig().getEntityID(), entity);  
 }  
  
 **return** entities;  
 }  
  
  
}

**package** com.mygdx.game.tools;  
  
**import** com.badlogic.gdx.math.GridPoint2;  
**import** com.badlogic.gdx.utils.Array;  
**import** com.badlogic.gdx.utils.ObjectMap;  
**import** com.mygdx.game.tools.Entity.AnimationType;  
**import** com.mygdx.game.inventory.InventoryItem.ItemNameID;  
  
  
**public class** EntityConfig {  
 **private** Array<AnimationConfig> **animationConfig**;  
 **private** Array<ItemNameID> **inventory**;  
 **private** Entity.State **state** = Entity.State.***IDLE***;  
 **private** Entity.Direction **direction** = Entity.Direction.***DOWN***;  
 **private** String **entityID**;  
 **private** String **conversationConfigPath**;  
 **private** String **questConfigPath**;  
 **private** String **currentQuestID**;  
 **private** String **itemNameID**;  
 **private** ObjectMap<String, String> **entityProperties**;  
  
 **public static enum** EntityProperties{  
 ***ENTITY\_HEALTH\_POINTS***,  
 ***ENTITY\_ATTACK\_POINTS***,  
 ***ENTITY\_DEFENSE\_POINTS***,  
 ***ENTITY\_HIT\_DAMAGE\_TOTAL***,  
 ***ENTITY\_XP\_REWARD***,  
 ***ENTITY\_GP\_REWARD***,  
 ***NONE*** }  
  
 **public** EntityConfig(){  
 **animationConfig** = **new** Array<AnimationConfig>();  
 **inventory** = **new** Array<ItemNameID>();  
 }  
  
 EntityConfig(EntityConfig config){  
 **state** = config.getState();  
 **direction** = config.getDirection();  
 **entityID** = config.getEntityID();  
 **itemNameID** = config.getItemNameID();  
  
 **animationConfig** = **new** Array<AnimationConfig>();  
 **animationConfig**.addAll(config.getAnimationConfig());  
  
 **inventory** = **new** Array<ItemNameID>();  
 **inventory**.addAll(config.getInventory());  
  
 **entityProperties** = **new** ObjectMap<String, String>();  
 **entityProperties**.putAll(config.**entityProperties**);  
 }  
  
 **public** ObjectMap<String, String> getEntityProperties() {  
 **return entityProperties**;  
 }  
  
 **public void** setEntityProperties(ObjectMap<String, String> entityProperties) {  
 **this**.**entityProperties** = entityProperties;  
 }  
  
 **public void** setPropertyValue(String key, String value){  
 **entityProperties**.put(key, value);  
 }  
  
 **public** String getPropertyValue(String key){  
 Object propertyVal = **entityProperties**.get(key);  
 **if**( propertyVal == **null** ) **return new** String();  
 **return** propertyVal.toString();  
 }  
  
 **public** String getEntityID() {  
 **return entityID**;  
 }  
  
 **public void** setEntityID(String entityID) {  
 **this**.**entityID** = entityID;  
 }  
  
 **public** Entity.Direction getDirection() {  
 **return direction**;  
 }  
  
 **public void** setDirection(Entity.Direction direction) {  
 **this**.**direction** = direction;  
 }  
  
 **public** Entity.State getState() {  
 **return state**;  
 }  
  
 **public void** setState(Entity.State state) {  
 **this**.**state** = state;  
 }  
  
 **public** String getItemNameID() {  
 **return itemNameID**;  
 }  
  
 **public void** setItemNameID(String itemNameID) {  
 **this**.**itemNameID** = itemNameID;  
 }  
  
 **public** Array<ItemNameID> getInventory() {  
 **return inventory**;  
 }  
  
 **public void** setInventory(Array<ItemNameID> inventory) {  
 **this**.**inventory** = inventory;  
 }  
  
 **public** Array<AnimationConfig> getAnimationConfig() {  
 **return animationConfig**;  
 }  
  
 **public void** addAnimationConfig(AnimationConfig animationConfig) {  
 **this**.**animationConfig**.add(animationConfig);  
 }  
  
 **public static class** AnimationConfig{  
 **private float frameDuration** = 1.0f;  
 **private** AnimationType **animationType**;  
 **private** Array<String> **texturePaths**;  
 **private** Array<GridPoint2> **gridPoints**;  
  
 **public** AnimationConfig(){  
 **animationType** = AnimationType.***IDLE***;  
 **texturePaths** = **new** Array<String>();  
 **gridPoints** = **new** Array<GridPoint2>();  
 }  
  
 **public float** getFrameDuration() {  
 **return frameDuration**;  
 }  
  
 **public void** setFrameDuration(**float** frameDuration) {  
 **this**.**frameDuration** = frameDuration;  
 }  
  
 **public** Array<String> getTexturePaths() {  
 **return texturePaths**;  
 }  
  
 **public void** setTexturePaths(Array<String> texturePaths) {  
 **this**.**texturePaths** = texturePaths;  
 }  
  
 **public** Array<GridPoint2> getGridPoints() {  
 **return gridPoints**;  
 }  
  
 **public void** setGridPoints(Array<GridPoint2> gridPoints) {  
 **this**.**gridPoints** = gridPoints;  
 }  
  
 **public** AnimationType getAnimationType() {  
 **return animationType**;  
 }  
  
 **public void** setAnimationType(AnimationType animationType) {  
 **this**.**animationType** = animationType;  
 }  
 }  
  
}

**package** com.mygdx.game.tools;  
  
  
**import** com.badlogic.gdx.utils.Json;  
**import** com.mygdx.game.components.Component;  
**import** com.mygdx.game.components.NPCGraphicsComponent;  
**import** com.mygdx.game.components.NPCInputComponent;  
**import** com.mygdx.game.components.NPCPhysicsComponent;  
**import** com.mygdx.game.components.PlayerGraphicsComponent;  
**import** com.mygdx.game.components.PlayerInputComponent;  
**import** com.mygdx.game.components.PlayerPhysicsComponent;  
  
**public class** EntityFactory {  
  
 **private static** Json *json* = **new** Json();  
  
 **public static enum** EntityType{  
 ***PLAYER***,  
 ***DEMO\_PLAYER***,  
 ***NPC*** }  
  
 **public static** String *PLAYER\_CONFIG* = **"json\_scripts/chara/hiro.json"**;  
  
 **public static** Entity getEntity(EntityType entityType){  
 Entity entity = **null**;  
 **switch**(entityType){  
 **case *PLAYER***:  
 entity = **new** Entity(**new** PlayerInputComponent(), **new** PlayerPhysicsComponent(), **new** PlayerGraphicsComponent());  
 entity.setEntityConfig(Entity.*getEntityConfig*(EntityFactory.*PLAYER\_CONFIG*));  
 entity.sendMessage(Component.MESSAGE.***LOAD\_ANIMATIONS***, *json*.toJson(entity.getEntityConfig()));  
 **return** entity;  
 **case *NPC***:  
 entity = **new** Entity(**new** NPCInputComponent(), **new** NPCPhysicsComponent(), **new** NPCGraphicsComponent());  
 **return** entity;  
 **default**:  
 **return null**;  
 }  
 }  
}

**package** com.mygdx.game.tools;  
  
*/\*\*  
 \* Created by Firas on 19/02/2017.  
 \*/***import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.assets.AssetManager;  
**import** com.badlogic.gdx.assets.loaders.MusicLoader;  
**import** com.badlogic.gdx.assets.loaders.SoundLoader;  
**import** com.badlogic.gdx.assets.loaders.TextureLoader;  
**import** com.badlogic.gdx.assets.loaders.resolvers.InternalFileHandleResolver;  
**import** com.badlogic.gdx.audio.Music;  
**import** com.badlogic.gdx.audio.Sound;  
**import** com.badlogic.gdx.graphics.Texture;  
**import** com.badlogic.gdx.graphics.g2d.TextureAtlas;  
**import** com.badlogic.gdx.maps.tiled.TiledMap;  
**import** com.badlogic.gdx.maps.tiled.TmxMapLoader;  
**import** com.badlogic.gdx.scenes.scene2d.ui.Skin;  
  
**public final class** Utility {  
  
 **public static final** AssetManager ***assetManager*** = **new** AssetManager();  
  
 **private static final** String ***TAG*** = Utility.**class**.getSimpleName();  
 *// a nice convenience class for managing file handles when resolving paths* **private static** InternalFileHandleResolver *filePathResolver* = **new** InternalFileHandleResolver();  
  
 **private final static** String ***GUI\_SHEET\_PATH*** = **"gui/gui\_sheet.atlas"**;  
 **private final static** String ***GUI\_SKINS\_PATH*** = **"gui/gui\_skins.json"**;  
 **private final static** String ***ITEMS\_SHEET\_PATH*** = **"items/small\_items\_sheet.txt"**;  
 **private final static** String ***HIRO\_SHEET\_PATH*** = **"sprites/chara/large\_hiro\_sheet.txt"**;  
  
 **private final static** String ***LARGE\_HIRAGANA\_SHEET\_PATH*** = **"japanese/hiragana/large\_hiragana\_sheet.txt"**;  
 **private final static** String ***LARGE\_KATAKANA\_SHEET\_PATH*** = **"japanese/katakana/large\_katakana\_sheet.txt"**;  
 **private final static** String ***LARGE\_ROMAJI\_SHEET\_PATH*** = **"japanese/romaji/large\_romaji\_sheet.txt"**;  
 **private final static** String ***MEDIUM\_HIRAGANA\_SHEET\_PATH*** = **"japanese/hiragana/medium\_hiragana\_sheet.txt"**;  
 **private final static** String ***MEDIUM\_KATAKANA\_SHEET\_PATH*** = **"japanese/katakana/medium\_katakana\_sheet.txt"**;  
 **private final static** String ***MEDIUM\_ROMAJI\_SHEET\_PATH*** = **"japanese/romaji/medium\_romaji\_sheet.txt"**;  
 **private final static** String ***SMALL\_HIRAGANA\_SHEET\_PATH*** = **"japanese/hiragana/small\_hiragana\_sheet.txt"**;  
 **private final static** String ***SMALL\_KATAKANA\_SHEET\_PATH*** = **"japanese/katakana/small\_katakana\_sheet.txt"**;  
 **private final static** String ***SMALL\_ROMAJI\_SHEET\_PATH*** = **"japanese/romaji/small\_romaji\_sheet.txt"**;  
  
 **private final static** String ***SMALL\_KANJI\_SHEET\_PATH*** = **"japanese/kanji/small\_kanji\_sheet.txt"**;  
 **private final static** String ***MEDIUM\_KANJI\_SHEET\_PATH*** = **"japanese/kanji/medium\_kanji\_sheet.txt"**;  
 **private final static** String ***LARGE\_KANJI\_SHEET\_PATH*** = **"japanese/kanji/large\_kanji\_sheet.txt"**;  
  
 **public static** TextureAtlas *GUI\_TEXTUREATLAS* = **new** TextureAtlas(***GUI\_SHEET\_PATH***);  
 **public static** TextureAtlas *ITEMS\_TEXTUREATLAS* = **new** TextureAtlas(***ITEMS\_SHEET\_PATH***);  
 **public static** TextureAtlas *HIRO\_TEXTUREATLAS* = **new** TextureAtlas(***HIRO\_SHEET\_PATH***);  
  
 **public static** TextureAtlas *LARGE\_HIRAGANA\_TEXTUREATLAS* = **new** TextureAtlas(***LARGE\_HIRAGANA\_SHEET\_PATH***);  
 **public static** TextureAtlas *LARGE\_KATAKANA\_TEXTUREATLAS* = **new** TextureAtlas(***LARGE\_KATAKANA\_SHEET\_PATH***);  
 **public static** TextureAtlas *LARGE\_ROMAJI\_TEXTUREATLAS* = **new** TextureAtlas(***LARGE\_ROMAJI\_SHEET\_PATH***);  
 **public static** TextureAtlas *MEDIUM\_HIRAGANA\_TEXTUREATLAS* = **new** TextureAtlas(***MEDIUM\_HIRAGANA\_SHEET\_PATH***);  
 **public static** TextureAtlas *MEDIUM\_KATAKANA\_TEXTUREATLAS* = **new** TextureAtlas(***MEDIUM\_KATAKANA\_SHEET\_PATH***);  
 **public static** TextureAtlas *MEDIUM\_ROMAJI\_TEXTUREATLAS* = **new** TextureAtlas(***MEDIUM\_ROMAJI\_SHEET\_PATH***);  
 **public static** TextureAtlas *SMALL\_HIRAGANA\_TEXTUREATLAS* = **new** TextureAtlas(***SMALL\_HIRAGANA\_SHEET\_PATH***);  
 **public static** TextureAtlas *SMALL\_KATAKANA\_TEXTUREATLAS* = **new** TextureAtlas(***SMALL\_KATAKANA\_SHEET\_PATH***);  
 **public static** TextureAtlas *SMALL\_ROMAJI\_TEXTUREATLAS* = **new** TextureAtlas(***SMALL\_ROMAJI\_SHEET\_PATH***);  
  
 **public static** TextureAtlas *SMALL\_KANJI\_TEXTUREATLAS* = **new** TextureAtlas(***SMALL\_KANJI\_SHEET\_PATH***);  
 **public static** TextureAtlas *MEDIUM\_KANJI\_TEXTUREATLAS* = **new** TextureAtlas(***MEDIUM\_KANJI\_SHEET\_PATH***);  
 **public static** TextureAtlas *LARGE\_KANJI\_TEXTUREATLAS* = **new** TextureAtlas(***LARGE\_KANJI\_SHEET\_PATH***);  
  
 **public static** Skin *GUI\_SKINS* = **new** Skin(Gdx.*files*.internal(***GUI\_SKINS\_PATH***), *GUI\_TEXTUREATLAS*);  
  
 **public static boolean** isAssetLoaded(String fileName){  
 **return *assetManager***.isLoaded(fileName);  
 }  
  
 **public static void** loadMusicAsset(String musicFilenamePath){  
 **if**( musicFilenamePath == **null** || musicFilenamePath.isEmpty() ){  
 **return**;  
 }  
  
 **if**( ***assetManager***.isLoaded(musicFilenamePath) ){  
 **return**;  
 }  
  
 *//load asset* **if**( *filePathResolver*.resolve(musicFilenamePath).exists() ){  
 ***assetManager***.setLoader(Music.**class**, **new** MusicLoader(*filePathResolver*));  
 ***assetManager***.load(musicFilenamePath, Music.**class**);  
 *//Until we add loading screen, just block until we load the map* ***assetManager***.finishLoadingAsset(musicFilenamePath);  
 Gdx.*app*.debug(***TAG***, **"Music loaded!: "** + musicFilenamePath);  
 }  
 **else**{  
 Gdx.*app*.debug(***TAG***, **"Music doesn't exist!: "** + musicFilenamePath );  
 }  
 }  
  
 **public static void** loadSoundAsset(String soundFilenamePath){  
 **if**( soundFilenamePath == **null** || soundFilenamePath.isEmpty() ){  
 **return**;  
 }  
  
 **if**( ***assetManager***.isLoaded(soundFilenamePath) ){  
 **return**;  
 }  
  
 *//load asset* **if**( *filePathResolver*.resolve(soundFilenamePath).exists() ){  
 ***assetManager***.setLoader(Sound.**class**, **new** SoundLoader(*filePathResolver*));  
 ***assetManager***.load(soundFilenamePath, Sound.**class**);  
 *//Until we add loading screen, just block until we load the map* ***assetManager***.finishLoadingAsset(soundFilenamePath);  
 Gdx.*app*.debug(***TAG***, **"Sound loaded!: "** + soundFilenamePath);  
 }  
 **else**{  
 Gdx.*app*.debug(***TAG***, **"Sound doesn't exist!: "** + soundFilenamePath );  
 }  
 }  
  
 **public static** Music getMusicAsset(String musicFilenamePath){  
 Music music = **null**;  
  
 *// once the asset manager is done loading* **if**( ***assetManager***.isLoaded(musicFilenamePath) ){  
 music = ***assetManager***.get(musicFilenamePath,Music.**class**);  
 } **else** {  
 Gdx.*app*.debug(***TAG***, **"Music is not loaded: "** + musicFilenamePath );  
 }  
  
 **return** music;  
 }  
  
 **public static** Sound getSoundAsset(String soundFilenamePath){  
 Sound sound = **null**;  
  
 *// once the asset manager is done loading* **if**( ***assetManager***.isLoaded(soundFilenamePath) ){  
 sound = ***assetManager***.get(soundFilenamePath,Sound.**class**);  
 } **else** {  
 Gdx.*app*.debug(***TAG***, **"Sound is not loaded: "** + soundFilenamePath );  
 }  
  
 **return** sound;  
 }  
  
 **public static void** loadMapAsset(String mapFilenamePath) {  
 **if** (mapFilenamePath == **null** || mapFilenamePath.isEmpty()) {  
 *//checks if null so to avoid exception in nxt statement* **return**;  
 }  
  
 **if**( ***assetManager***.isLoaded(mapFilenamePath) ){  
 **return**;  
 }  
  
 *//load asset* **if** (*filePathResolver*.resolve(mapFilenamePath).exists()) {  
 ***assetManager***.setLoader(TiledMap.**class**, **new** TmxMapLoader(*filePathResolver*));  
 ***assetManager***.load(mapFilenamePath, TiledMap.**class**);  
 *//until loading screen is added, block* ***assetManager***.finishLoadingAsset(mapFilenamePath);  
 Gdx.*app*.debug(***TAG***, **"Map loaded: "** + mapFilenamePath);  
 } **else**{  
 Gdx.*app*.debug(***TAG***, **"Map doesn't exist: "** + mapFilenamePath);  
 }  
 }  
  
 **public static** TiledMap getMapAsset(String mapFilenamePath){  
 TiledMap map = **null**;  
 *//checks null or not so to avoid exception in nxt statement* **if**(***assetManager***.isLoaded(mapFilenamePath)) {  
 map = ***assetManager***.get(mapFilenamePath, TiledMap.**class**);  
 } **else** {  
 Gdx.*app*.debug(***TAG***, **"Map is not loaded: "** + mapFilenamePath);  
 }  
 **return** map;  
 }  
  
 **public static void** loadTextureAsset(String textureFilenamePath){  
 **if**(textureFilenamePath == **null** || textureFilenamePath.isEmpty()){  
 *//checks if null so to avoid exception in nxt statement* **return**;  
 }  
 *//load asset* **if**(*filePathResolver*.resolve(textureFilenamePath).exists()){  
 ***assetManager***.setLoader(Texture.**class**, **new** TextureLoader(*filePathResolver*));  
 ***assetManager***.load(textureFilenamePath, Texture.**class**);  
 *//until loading screen is added, block* ***assetManager***.finishLoadingAsset(textureFilenamePath);  
 *//Gdx.app.debug(TAG, "Texture loaded: " + textureFilenamePath);* } **else**{  
 Gdx.*app*.debug(***TAG***, **"Texture doesn't exist: "** + textureFilenamePath);  
 }  
 }  
  
 **public static** Texture getTextureAsset(String textureFilenamePath){  
 Texture texture = **null**;  
  
 *//once asset manger is done loading* **if**(***assetManager***.isLoaded(textureFilenamePath)){  
 texture = ***assetManager***.get(textureFilenamePath, Texture.**class**);  
 } **else** {  
 Gdx.*app*.debug(***TAG***, **"Texture is not loaded: "** + textureFilenamePath);  
 }  
 **return** texture;  
 }  
  
  
}

**package** com.mygdx.game;  
  
**import** com.badlogic.gdx.Game;  
**import** com.badlogic.gdx.Screen;  
**import** com.mygdx.game.screens.GameOverScreen;  
**import** com.mygdx.game.screens.LoadGameScreen;  
**import** com.mygdx.game.screens.MainGameScreen;  
**import** com.mygdx.game.screens.MainMenuScreen;  
**import** com.mygdx.game.screens.NewGameScreen;  
  
**public class** FinalKanjiQuest **extends** Game {  
  
 **private static** MainGameScreen *mainGameScreen*;  
 **private static** MainMenuScreen *mainMenuScreen*;  
 **private static** LoadGameScreen *loadGameScreen*;  
 **private static** NewGameScreen *newGameScreen*;  
 **private static** GameOverScreen *gameOverScreen*;  
  
 **public static enum** ScreenType{  
 ***MainMenu***,  
 ***MainGame***,  
 ***LoadGame***,  
 ***NewGame***,  
 ***GameOver*** }  
  
 **public** Screen getScreenType(ScreenType screenType){  
 **switch**(screenType){  
 **case *MainMenu***:  
 **return** *mainMenuScreen*;  
 **case *MainGame***:  
 **return** *mainGameScreen*;  
 **case *LoadGame***:  
 **return** *loadGameScreen*;  
 **case *NewGame***:  
 **return** *newGameScreen*;  
 **case *GameOver***:  
 **return** *gameOverScreen*;  
 **default**:  
 **return** *mainMenuScreen*;  
 }  
  
 }  
  
 @Override  
 **public void** create(){  
 *mainGameScreen* = **new** MainGameScreen(**this**);  
 *mainMenuScreen* = **new** MainMenuScreen(**this**);  
 *loadGameScreen* = **new** LoadGameScreen(**this**);  
 *newGameScreen* = **new** NewGameScreen(**this**);  
 *gameOverScreen* = **new** GameOverScreen(**this**);  
 setScreen(*mainMenuScreen*);  
 }  
  
 @Override  
 **public void** dispose(){  
 *mainGameScreen*.dispose();  
 *mainMenuScreen*.dispose();  
 *loadGameScreen*.dispose();  
 *newGameScreen*.dispose();  
 *gameOverScreen*.dispose();  
 }  
  
}

## src.com.mygdx.game.desktop

**package** com.mygdx.game.desktop;  
  
**import** com.badlogic.gdx.backends.lwjgl.LwjglApplication;  
**import** com.badlogic.gdx.backends.lwjgl.LwjglApplicationConfiguration;  
**import** com.badlogic.gdx.Application;  
**import** com.badlogic.gdx.Gdx;  
**import** com.mygdx.game.FinalKanjiQuest;  
  
  
**public class** DesktopLauncher {  
 **public static void** main (String[] arg) {  
 LwjglApplicationConfiguration config = **new** LwjglApplicationConfiguration();  
  
 config.**title** = **"FinalKanjiQuest"**;  
 *// set to false so that we use the much more stable and mature implementation of OpenGL ES* config.**useGL30** = **false**;  
 config.**width** = 800;  
 config.**height** = 600;  
  
 Application app = **new** LwjglApplication(**new** FinalKanjiQuest(),config);  
  
 Gdx.*app* = app;  
 *// logging level that displays all messages* Gdx.*app*.setLogLevel(Application.***LOG\_DEBUG***);  
 *//Gdx.app.setLogLevel(Application.LOG\_INFO);  
 // Gdx.app.setLogLevel(Application.LOG\_ERROR);  
 // Gdx.app.setLogLevel(Application.LOG\_NONE);* }  
}

## json\_scripts

{  
entityID : BLACKSMITH  
state : IMMOBILE  
direction : DOWN  
animationConfig: [  
 {  
 frameDuration: 1.0  
 animationType: IMMOBILE  
 texturePaths: [  
 sprites/chara/blacksmith\_sheet.png  
 ]  
 gridPoints: [  
 {  
 x: 0  
 y: 0  
 }  
 {  
 x: 1  
 y: 0  
 }  
 {  
 x: 2  
 y: 0  
 }  
 ]  
 }   
]  
}

{  
entityID : HERBALIST  
state : IMMOBILE  
direction : DOWN  
animationConfig: [  
 {  
 frameDuration: 1.0  
 animationType: IMMOBILE  
 texturePaths: [  
 sprites/chara/herbalist\_sheet.png  
 ]  
 gridPoints: [  
 {  
 x: 0  
 y: 0  
 }  
 {  
 x: 1  
 y: 0  
 }  
 {  
 x: 2  
 y: 0  
 }  
 ]  
 }   
]  
}

{  
entityID : PLAYER\_START  
state : IDLE  
direction : DOWN  
inventory : [  
 {  
 value: POTION\_BLUE  
 },  
 {  
 value: POTION\_BLUE  
 },  
 {  
 value: POTION\_BLUE  
 },  
 {  
 value: POTION\_BLUE  
 },  
 {  
 value: POTION\_BLUE  
 },  
 {  
 value: MEDIUM\_POTION\_BLUE  
 },  
 {  
 value: MEDIUM\_POTION\_BLUE  
 },  
 {  
 value: MEDIUM\_POTION\_BLUE  
 },  
 {  
 value: MEDIUM\_POTION\_BLUE  
 },  
 {  
 value: MEDIUM\_POTION\_BLUE  
 },  
 {  
 value: LARGE\_POTION\_BLUE  
 },  
 {  
 value: LARGE\_POTION\_BLUE  
 },  
 {  
 value: LARGE\_POTION\_BLUE  
 },  
 {  
 value: LARGE\_POTION\_BLUE  
 },  
 {  
 value: LARGE\_POTION\_BLUE  
 },  
 {  
 value: HEART  
 },  
 {  
 value: HEART  
 },  
 {  
 value: HEART  
 },  
 {  
 value: HEART  
 },  
 {  
 value: HEART  
 },  
 {  
 value: H\_RUNESTONE  
 },  
 {  
 value: H\_RUNESTONE  
 },  
 {  
 value: H\_RUNESTONE  
 },  
 {  
 value: NEGATIVE\_H\_RUNESTONE  
 },  
 {  
 value: NEGATIVE\_H\_RUNESTONE  
 },  
 {  
 value: NEGATIVE\_H\_RUNESTONE  
 },  
 {  
 value: K\_RUNESTONE  
 },  
 {  
 value: K\_RUNESTONE  
 },  
 {  
 value: K\_RUNESTONE  
 },  
 {  
 value: NEGATIVE\_K\_RUNESTONE  
 },  
 {  
 value: NEGATIVE\_K\_RUNESTONE  
 },  
 {  
 value: NEGATIVE\_K\_RUNESTONE  
 },  
]  
animationConfig: [  
 {  
 frameDuration: 0.25  
 animationType: WALK\_DOWN  
 texturePaths: [  
 sprites/chara/hiro\_sheet.png  
 ]  
 gridPoints: [  
 {  
 x: 0  
 y: 0  
 }  
   
 {  
 x: 0  
 y: 1  
 }  
 {  
 x: 0  
 y: 2  
 }  
 {  
 x: 0  
 y: 3  
 }  
 ]  
 }  
 {  
 frameDuration: 0.25  
 animationType: WALK\_DOWN\_LEFT  
 texturePaths: [  
 sprites/chara/hiro\_sheet.png  
 ]  
 gridPoints: [  
 {  
 x: 1  
 y: 0  
 }  
  
 {  
 x: 1  
 y: 1  
 }  
 {  
 x: 1  
 y: 2  
 }  
 {  
 x: 1  
 y: 3  
 }  
 ]  
 }  
 {  
 frameDuration: 0.25  
 animationType: WALK\_DOWN\_RIGHT  
 texturePaths: [  
 sprites/chara/hiro\_sheet.png  
 ]  
 gridPoints: [  
 {  
 x: 2  
 y: 0  
 }  
  
 {  
 x: 2  
 y: 1  
 }  
 {  
 x: 2  
 y: 2  
 }  
 {  
 x: 2  
 y: 3  
 }  
 ]  
 }  
 {  
 frameDuration: 0.25  
 animationType: WALK\_LEFT  
 texturePaths: [  
 sprites/chara/hiro\_sheet.png  
 ]  
 gridPoints: [  
 {  
 x: 3  
 y: 0  
 }  
   
 {  
 x: 3  
 y: 1  
 }  
 {  
 x: 3  
 y: 2  
 }  
 {  
 x: 3  
 y: 3  
 }  
 ]  
 }  
 {  
 frameDuration: 0.25  
 animationType: WALK\_RIGHT  
 texturePaths: [  
 sprites/chara/hiro\_sheet.png  
 ]  
 gridPoints: [  
 {  
 x: 4  
 y: 0  
 }  
   
 {  
 x: 4  
 y: 1  
 }  
 {  
 x: 4  
 y: 2  
 }  
 {  
 x: 4  
 y: 3  
 }  
 ]  
 }  
 {  
 frameDuration: 0.25  
 animationType: WALK\_UP  
 texturePaths: [  
 sprites/chara/hiro\_sheet.png  
 ]  
 gridPoints: [  
 {  
 x: 5  
 y: 0  
 }  
   
 {  
 x: 5  
 y: 1  
 }  
 {  
 x: 5  
 y: 2  
 }  
 {  
 x: 5  
 y: 3  
 }  
 ]  
 }  
 {  
 frameDuration: 0.25  
 animationType: WALK\_UP\_LEFT  
 texturePaths: [  
 sprites/chara/hiro\_sheet.png  
 ]  
 gridPoints: [  
 {  
 x: 6  
 y: 0  
 }  
  
 {  
 x: 6  
 y: 1  
 }  
 {  
 x: 6  
 y: 2  
 }  
 {  
 x: 6  
 y: 3  
 }  
 ]  
 }  
 {  
 frameDuration: 0.25  
 animationType: WALK\_UP\_RIGHT  
 texturePaths: [  
 sprites/chara/hiro\_sheet.png  
 ]  
 gridPoints: [  
 {  
 x: 7  
 y: 0  
 }  
  
 {  
 x: 7  
 y: 1  
 }  
 {  
 x: 7  
 y: 2  
 }  
 {  
 x: 7  
 y: 3  
 }  
 ]  
 }  
]  
}

{  
entityID : INN\_CLARK  
state : IDLE  
direction : DOWN  
animationConfig: [  
 {  
 frameDuration: 0.25  
 animationType: WALK\_DOWN  
 texturePaths: [  
 sprites/chara/inn\_clark\_sheet.png  
 ]  
 gridPoints: [  
 {  
 x: 0  
 y: 0  
 }  
   
 {  
 x: 0  
 y: 1  
 }  
 {  
 x: 0  
 y: 2  
 }  
 {  
 x: 0  
 y: 3  
 }  
 ]  
 }  
 {  
 frameDuration: 0.25  
 animationType: WALK\_LEFT  
 texturePaths: [  
 sprites/chara/inn\_clark\_sheet.png  
 ]  
 gridPoints: [  
 {  
 x: 1  
 y: 0  
 }  
   
 {  
 x: 1  
 y: 1  
 }  
 {  
 x: 1  
 y: 2  
 }  
 {  
 x: 1  
 y: 3  
 }  
 ]  
 }  
 {  
 frameDuration: 0.25  
 animationType: WALK\_RIGHT  
 texturePaths: [  
 sprites/chara/inn\_clark\_sheet.png  
 ]  
 gridPoints: [  
 {  
 x: 2  
 y: 0  
 }  
   
 {  
 x: 2  
 y: 1  
 }  
 {  
 x: 2  
 y: 2  
 }  
 {  
 x: 2  
 y: 3  
 }  
 ]  
 }  
 {  
 frameDuration: 0.25  
 animationType: WALK\_UP  
 texturePaths: [  
 sprites/chara/inn\_clark\_sheet.png  
 ]  
 gridPoints: [  
 {  
 x: 3  
 y: 0  
 }  
   
 {  
 x: 3  
 y: 1  
 }  
 {  
 x: 3  
 y: 2  
 }  
 {  
 x: 3  
 y: 3  
 }  
 ]  
 }   
]  
}

{  
entityID : LIBRARIAN  
state : IMMOBILE  
direction : DOWN  
animationConfig: [  
 {  
 frameDuration: 1.0  
 animationType: IMMOBILE  
 texturePaths: [  
 sprites/chara/librarian\_sheet.png  
 ]  
 gridPoints: [  
 {  
 x: 0  
 y: 0  
 }  
 {  
 x: 1  
 y: 0  
 }  
 {  
 x: 2  
 y: 0  
 }  
 ]  
 }   
]  
}

{  
entityID : WALKAROUND\_FOUR  
state : IDLE  
direction : DOWN  
animationConfig: [  
 {  
 frameDuration: 0.25  
 animationType: WALK\_DOWN  
 texturePaths: [  
 sprites/chara/walkaround\_four\_sheet.png  
 ]  
 gridPoints: [  
 {  
 x: 0  
 y: 0  
 }  
   
 {  
 x: 0  
 y: 1  
 }  
 {  
 x: 0  
 y: 2  
 }  
 {  
 x: 0  
 y: 3  
 }  
 ]  
 }  
 {  
 frameDuration: 0.25  
 animationType: WALK\_LEFT  
 texturePaths: [  
 sprites/chara/walkaround\_four\_sheet.png  
 ]  
 gridPoints: [  
 {  
 x: 1  
 y: 0  
 }  
   
 {  
 x: 1  
 y: 1  
 }  
 {  
 x: 1  
 y: 2  
 }  
 {  
 x: 1  
 y: 3  
 }  
 ]  
 }  
 {  
 frameDuration: 0.25  
 animationType: WALK\_RIGHT  
 texturePaths: [  
 sprites/chara/walkaround\_four\_sheet.png  
 ]  
 gridPoints: [  
 {  
 x: 2  
 y: 0  
 }  
   
 {  
 x: 2  
 y: 1  
 }  
 {  
 x: 2  
 y: 2  
 }  
 {  
 x: 2  
 y: 3  
 }  
 ]  
 }  
 {  
 frameDuration: 0.25  
 animationType: WALK\_UP  
 texturePaths: [  
 sprites/chara/walkaround\_four\_sheet.png  
 ]  
 gridPoints: [  
 {  
 x: 3  
 y: 0  
 }  
   
 {  
 x: 3  
 y: 1  
 }  
 {  
 x: 3  
 y: 2  
 }  
 {  
 x: 3  
 y: 3  
 }  
 ]  
 }   
]  
}

{  
entityID : WALKAROUND\_ONE  
state : IDLE  
direction : DOWN  
animationConfig: [  
 {  
 frameDuration: 0.25  
 animationType: WALK\_DOWN  
 texturePaths: [  
 sprites/chara/walkaround\_one\_sheet.png  
 ]  
 gridPoints: [  
 {  
 x: 0  
 y: 0  
 }  
   
 {  
 x: 0  
 y: 1  
 }  
 {  
 x: 0  
 y: 2  
 }  
 {  
 x: 0  
 y: 3  
 }  
 ]  
 }  
 {  
 frameDuration: 0.25  
 animationType: WALK\_LEFT  
 texturePaths: [  
 sprites/chara/walkaround\_one\_sheet.png  
 ]  
 gridPoints: [  
 {  
 x: 1  
 y: 0  
 }  
   
 {  
 x: 1  
 y: 1  
 }  
 {  
 x: 1  
 y: 2  
 }  
 {  
 x: 1  
 y: 3  
 }  
 ]  
 }  
 {  
 frameDuration: 0.25  
 animationType: WALK\_RIGHT  
 texturePaths: [  
 sprites/chara/walkaround\_one\_sheet.png  
 ]  
 gridPoints: [  
 {  
 x: 2  
 y: 0  
 }  
   
 {  
 x: 2  
 y: 1  
 }  
 {  
 x: 2  
 y: 2  
 }  
 {  
 x: 2  
 y: 3  
 }  
 ]  
 }  
 {  
 frameDuration: 0.25  
 animationType: WALK\_UP  
 texturePaths: [  
 sprites/chara/walkaround\_one\_sheet.png  
 sprites/chara/walkaround\_one\_sheet.png  
 ]  
 gridPoints: [  
 {  
 x: 3  
 y: 0  
 }  
   
 {  
 x: 3  
 y: 1  
 }  
 {  
 x: 3  
 y: 2  
 }  
 {  
 x: 3  
 y: 3  
 }  
 ]  
 }   
]  
}

{  
entityID : WALKAROUND\_THREE  
state : IDLE  
direction : DOWN  
animationConfig: [  
 {  
 frameDuration: 0.25  
 animationType: WALK\_DOWN  
 texturePaths: [  
 sprites/chara/walkaround\_three\_sheet.png  
 ]  
 gridPoints: [  
 {  
 x: 0  
 y: 0  
 }  
   
 {  
 x: 0  
 y: 1  
 }  
 {  
 x: 0  
 y: 2  
 }  
 {  
 x: 0  
 y: 3  
 }  
 ]  
 }  
 {  
 frameDuration: 0.25  
 animationType: WALK\_LEFT  
 texturePaths: [  
 sprites/chara/walkaround\_three\_sheet.png  
 ]  
 gridPoints: [  
 {  
 x: 1  
 y: 0  
 }  
   
 {  
 x: 1  
 y: 1  
 }  
 {  
 x: 1  
 y: 2  
 }  
 {  
 x: 1  
 y: 3  
 }  
 ]  
 }  
 {  
 frameDuration: 0.25  
 animationType: WALK\_RIGHT  
 texturePaths: [  
 sprites/chara/walkaround\_three\_sheet.png  
 ]  
 gridPoints: [  
 {  
 x: 2  
 y: 0  
 }  
   
 {  
 x: 2  
 y: 1  
 }  
 {  
 x: 2  
 y: 2  
 }  
 {  
 x: 2  
 y: 3  
 }  
 ]  
 }  
 {  
 frameDuration: 0.25  
 animationType: WALK\_UP  
 texturePaths: [  
 sprites/chara/walkaround\_three\_sheet.png  
 ]  
 gridPoints: [  
 {  
 x: 3  
 y: 0  
 }  
   
 {  
 x: 3  
 y: 1  
 }  
 {  
 x: 3  
 y: 2  
 }  
 {  
 x: 3  
 y: 3  
 }  
 ]  
 }   
]  
}

{  
entityID : WALKAROUND\_TWO  
state : IDLE  
direction : DOWN  
animationConfig: [  
 {  
 frameDuration: 0.25  
 animationType: WALK\_DOWN  
 texturePaths: [  
 sprites/chara/walkaround\_two\_sheet.png  
 ]  
 gridPoints: [  
 {  
 x: 0  
 y: 0  
 }  
   
 {  
 x: 0  
 y: 1  
 }  
 {  
 x: 0  
 y: 2  
 }  
 {  
 x: 0  
 y: 3  
 }  
 ]  
 }  
 {  
 frameDuration: 0.25  
 animationType: WALK\_LEFT  
 texturePaths: [  
 sprites/chara/walkaround\_two\_sheet.png  
 ]  
 gridPoints: [  
 {  
 x: 1  
 y: 0  
 }  
   
 {  
 x: 1  
 y: 1  
 }  
 {  
 x: 1  
 y: 2  
 }  
 {  
 x: 1  
 y: 3  
 }  
 ]  
 }  
 {  
 frameDuration: 0.25  
 animationType: WALK\_RIGHT  
 texturePaths: [  
 sprites/chara/walkaround\_two\_sheet.png  
 ]  
 gridPoints: [  
 {  
 x: 2  
 y: 0  
 }  
   
 {  
 x: 2  
 y: 1  
 }  
 {  
 x: 2  
 y: 2  
 }  
 {  
 x: 2  
 y: 3  
 }  
 ]  
 }  
 {  
 frameDuration: 0.25  
 animationType: WALK\_UP  
 texturePaths: [  
 sprites/chara/walkaround\_two\_sheet.png  
 ]  
 gridPoints: [  
 {  
 x: 3  
 y: 0  
 }  
   
 {  
 x: 3  
 y: 1  
 }  
 {  
 x: 3  
 y: 2  
 }  
 {  
 x: 3  
 y: 3  
 }  
 ]  
 }   
]  
}

[  
  
{  
 zoneID : 1  
 monsters: .kanjiHa.kanjiHana.kanjiKi.kanjiMori.kanjiMushi.  
},  
{  
 zoneID : 2  
 monsters: .kanjiIshi.kanjiKaze.kanjiTsuchi.kanjiUchi.kanjiYama.  
},  
{  
 zoneID : 3  
 monsters: .kanjiUmi.kanjiSakana.kanjiMizu.kanjiKai.kanjiIke.  
},  
{  
 zoneID : 4  
 monsters: .kanjiYuki.kanjiKoori.kanjiKawa.kanjiHayashi.kanjiFuyu.  
},  
{  
 zoneID : 5  
 monsters: .kanjiTo.kanjiMon.kanjiTera.kanjiAna.kanjiSoto.  
},  
{  
 zoneID : 6  
 monsters: .kanjiKa.kanjiHi.kanjiChi.kanjiAka.kanjiHikari.  
},  
  
]

[  
{  
 itemAttributes: 1  
 itemUseType: 1  
 itemNameID: POTION\_BLUE  
 itemShortDescription: A small potion that recovers 1 heart  
 itemUseValue: 1  
},  
{  
 itemAttributes: 1  
 itemUseType: 1  
 itemNameID: MEDIUM\_POTION\_BLUE  
 itemShortDescription: A medium potion that recovers 3 hearts  
 itemUseValue: 3  
},  
{  
 itemAttributes: 1  
 itemUseType: 1  
 itemNameID: LARGE\_POTION\_BLUE  
 itemShortDescription: A Large potion that recovers 5 hearts  
 itemUseValue: 5  
},  
{  
 itemAttributes: 1  
 itemUseType: 2  
 itemNameID: HEART  
 itemShortDescription: Grants the user an extra health-heart  
 itemUseValue: 0  
},  
{  
 itemAttributes: 1  
 itemUseType: 4  
 itemNameID: H\_RUNESTONE  
 itemShortDescription: A mysterious stone that stops encounters with hiragana  
 itemUseValue: 0  
},  
{  
 itemAttributes: 1  
 itemUseType: 8  
 itemNameID: NEGATIVE\_H\_RUNESTONE  
 itemShortDescription: A mysterious stone that resets encounters with hiragana  
 itemUseValue: 0  
},  
{  
 itemAttributes: 1  
 itemUseType: 16  
 itemNameID: K\_RUNESTONE  
 itemShortDescription: A mysterious stone that stops encounters with katakana  
 itemUseValue: 0  
},  
{  
 itemAttributes: 1  
 itemUseType: 32  
 itemNameID: NEGATIVE\_K\_RUNESTONE  
 itemShortDescription: A mysterious stone that resets encounters with katakana  
 itemUseValue: 0  
},  
  
]

[  
 {  
 hiraganaEquivalent: hiraganaA  
 katakanaEquivalent: katakanaA  
 romajiEquivalent: romajiA  
 },  
 {  
 hiraganaEquivalent: hiraganaI  
 katakanaEquivalent: katakanaI  
 romajiEquivalent: romajiI  
 },  
 {  
 hiraganaEquivalent: hiraganaU  
 katakanaEquivalent: katakanaU  
 romajiEquivalent: romajiU  
 },  
 {  
 hiraganaEquivalent: hiraganaE  
 katakanaEquivalent: katakanaE  
 romajiEquivalent: romajiE  
 },  
 {  
 hiraganaEquivalent: hiraganaO  
 katakanaEquivalent: katakanaO  
 romajiEquivalent: romajiO  
 },  
 {  
 hiraganaEquivalent: hiraganaKa  
 katakanaEquivalent: katakanaKa  
 romajiEquivalent: romajiKa  
 },  
 {  
 hiraganaEquivalent: hiraganaKi  
 katakanaEquivalent: katakanaKi  
 romajiEquivalent: romajiKi  
 },  
 {  
 hiraganaEquivalent: hiraganaKu  
 katakanaEquivalent: katakanaKu  
 romajiEquivalent: romajiKu  
 },  
 {  
 hiraganaEquivalent: hiraganaKe  
 katakanaEquivalent: katakanaKe  
 romajiEquivalent: romajiKe  
 },  
 {  
 hiraganaEquivalent: hiraganaKo  
 katakanaEquivalent: katakanaKo  
 romajiEquivalent: romajiKo  
 },  
 {  
 hiraganaEquivalent: hiraganaSa  
 katakanaEquivalent: katakanaSa  
 romajiEquivalent: romajiSa  
 },  
 {  
 hiraganaEquivalent: hiraganaShi  
 katakanaEquivalent: katakanaShi  
 romajiEquivalent: romajiShi  
 },  
 {  
 hiraganaEquivalent: hiraganaSu  
 katakanaEquivalent: katakanaSu  
 romajiEquivalent: romajiSu  
 },  
 {  
 hiraganaEquivalent: hiraganaSe  
 katakanaEquivalent: katakanaSe  
 romajiEquivalent: romajiSe  
 },  
 {  
 hiraganaEquivalent: hiraganaSo  
 katakanaEquivalent: katakanaSo  
 romajiEquivalent: romajiSo  
 },  
 {  
 hiraganaEquivalent: hiraganaTa  
 katakanaEquivalent: katakanaTa  
 romajiEquivalent: romajiTa  
 },  
 {  
 hiraganaEquivalent: hiraganaChi  
 katakanaEquivalent: katakanaChi  
 romajiEquivalent: romajiChi  
 },  
 {  
 hiraganaEquivalent: hiraganaTsu  
 katakanaEquivalent: katakanaTsu  
 romajiEquivalent: romajiTsu  
 },  
 {  
 hiraganaEquivalent: hiraganaTe  
 katakanaEquivalent: katakanaTe  
 romajiEquivalent: romajiTe  
 },  
 {  
 hiraganaEquivalent: hiraganaTo  
 katakanaEquivalent: katakanaTo  
 romajiEquivalent: romajiTo  
 },  
 {  
 hiraganaEquivalent: hiraganaNa  
 katakanaEquivalent: katakanaNa  
 romajiEquivalent: romajiNa  
 },  
 {  
 hiraganaEquivalent: hiraganaNi  
 katakanaEquivalent: katakanaN  
 romajiEquivalent: romajiNi  
 },  
 {  
 hiraganaEquivalent: hiraganaNu  
 katakanaEquivalent: katakanaNu  
 romajiEquivalent: romajiNu  
 },  
 {  
 hiraganaEquivalent: hiraganaNe  
 katakanaEquivalent: katakanaNe  
 romajiEquivalent: romajiNe  
 },  
 {  
 hiraganaEquivalent: hiraganaNo  
 katakanaEquivalent: katakanaNo  
 romajiEquivalent: romajiNo  
 },  
 {  
 hiraganaEquivalent: hiraganaHa  
 katakanaEquivalent: katakanaHa  
 romajiEquivalent: romajiHa  
 },  
 {  
 hiraganaEquivalent: hiraganaHi  
 katakanaEquivalent: katakanaHi  
 romajiEquivalent: romajiHi  
 },  
 {  
 hiraganaEquivalent: hiraganaHu  
 katakanaEquivalent: katakanaHu  
 romajiEquivalent: romajiHu  
 },  
 {  
 hiraganaEquivalent: hiraganaHe  
 katakanaEquivalent: katakanaHe  
 romajiEquivalent: romajiHe  
 },  
 {  
 hiraganaEquivalent: hiraganaHo  
 katakanaEquivalent: katakanaHo  
 romajiEquivalent: romajiHo  
 },  
 {  
 hiraganaEquivalent: hiraganaMa  
 katakanaEquivalent: katakanaMa  
 romajiEquivalent: romajiMa  
 },  
 {  
 hiraganaEquivalent: hiraganaMi  
 katakanaEquivalent: katakanaMi  
 romajiEquivalent: romajiMi  
 },  
 {  
 hiraganaEquivalent: hiraganaMu  
 katakanaEquivalent: katakanaMu  
 romajiEquivalent: romajiMu  
 },  
 {  
 hiraganaEquivalent: hiraganaMe  
 katakanaEquivalent: katakanaMe  
 romajiEquivalent: romajiMe  
 },  
 {  
 hiraganaEquivalent: hiraganaMo  
 katakanaEquivalent: katakanaMo  
 romajiEquivalent: romajiMo  
 },  
 {  
 hiraganaEquivalent: hiraganaRa  
 katakanaEquivalent: katakanaRa  
 romajiEquivalent: romajiRa  
 },  
 {  
 hiraganaEquivalent: hiraganaRi  
 katakanaEquivalent: katakanaRi  
 romajiEquivalent: romajiRi  
 },  
 {  
 hiraganaEquivalent: hiraganaRu  
 katakanaEquivalent: katakanaRu  
 romajiEquivalent: romajiRu  
 },  
 {  
 hiraganaEquivalent: hiraganaRe  
 katakanaEquivalent: katakanaRe  
 romajiEquivalent: romajiRe  
 },  
 {  
 hiraganaEquivalent: hiraganaRo  
 katakanaEquivalent: katakanaRo  
 romajiEquivalent: romajiRo  
 },  
 {  
 hiraganaEquivalent: hiraganaYa  
 katakanaEquivalent: katakanaYa  
 romajiEquivalent: romajiYa  
 },  
 {  
 hiraganaEquivalent: hiraganaYu  
 katakanaEquivalent: katakanaYu  
 romajiEquivalent: romajiYu  
 },  
 {  
 hiraganaEquivalent: hiraganaYo  
 katakanaEquivalent: katakanaYo  
 romajiEquivalent: romajiYo  
 },  
 {  
 hiraganaEquivalent: hiraganaWa  
 katakanaEquivalent: katakanaWa  
 romajiEquivalent: romajiWa  
 },  
 {  
 hiraganaEquivalent: hiraganaWo  
 katakanaEquivalent: katakanaWo  
 romajiEquivalent: romajiWo  
 },  
 {  
 hiraganaEquivalent: hiraganaN  
 katakanaEquivalent: katakanaN  
 romajiEquivalent: romajiN  
 },  
 {  
 hiraganaEquivalent: hiraganaGa  
 katakanaEquivalent: katakanaGa  
 romajiEquivalent: romajiGa  
 },  
 {  
 hiraganaEquivalent: hiraganaGi  
 katakanaEquivalent: katakanaGi  
 romajiEquivalent: romajiGi  
 },  
 {  
 hiraganaEquivalent: hiraganaGu  
 katakanaEquivalent: katakanaGu  
 romajiEquivalent: romajiGu  
 },  
 {  
 hiraganaEquivalent: hiraganaGe  
 katakanaEquivalent: katakanaGe  
 romajiEquivalent: romajiGe  
 },  
 {  
 hiraganaEquivalent: hiraganaGo  
 katakanaEquivalent: katakanaGo  
 romajiEquivalent: romajiGo  
 },  
 {  
 hiraganaEquivalent: hiraganaZa  
 katakanaEquivalent: katakanaZa  
 romajiEquivalent: romajiZa  
 },  
 {  
 hiraganaEquivalent: hiraganaJi  
 katakanaEquivalent: katakanaJi  
 romajiEquivalent: romajiJi  
 },  
 {  
 hiraganaEquivalent: hiraganaZu  
 katakanaEquivalent: katakanaZu  
 romajiEquivalent: romajiZu  
 },  
 {  
 hiraganaEquivalent: hiraganaZe  
 katakanaEquivalent: katakanaZe  
 romajiEquivalent: romajiZe  
 },  
 {  
 hiraganaEquivalent: hiraganaZo  
 katakanaEquivalent: katakanaZo  
 romajiEquivalent: romajiZo  
 },  
 {  
 hiraganaEquivalent: hiraganaDa  
 katakanaEquivalent: katakanaDa  
 romajiEquivalent: romajiDa  
 },  
 {  
 hiraganaEquivalent: hiraganaDi  
 katakanaEquivalent: katakanaDi  
 romajiEquivalent: romajiDi  
 },  
 {  
 hiraganaEquivalent: hiraganaDu  
 katakanaEquivalent: katakanaDu  
 romajiEquivalent: romajiDu  
 },  
 {  
 hiraganaEquivalent: hiraganaDe  
 katakanaEquivalent: katakanaDe  
 romajiEquivalent: romajiDe  
 },  
 {  
 hiraganaEquivalent: hiraganaDo  
 katakanaEquivalent: katakanaDo  
 romajiEquivalent: romajiDo  
 },  
 {  
 hiraganaEquivalent: hiraganaBa  
 katakanaEquivalent: katakanaBa  
 romajiEquivalent: romajiBa  
 },  
 {  
 hiraganaEquivalent: hiraganaBi  
 katakanaEquivalent: katakanaBi  
 romajiEquivalent: romajiBi  
 },  
 {  
 hiraganaEquivalent: hiraganaBu  
 katakanaEquivalent: katakanaBu  
 romajiEquivalent: romajiBu  
 },  
 {  
 hiraganaEquivalent: hiraganaBe  
 katakanaEquivalent: katakanaBe  
 romajiEquivalent: romajiBe  
 },  
 {  
 hiraganaEquivalent: hiraganaBo  
 katakanaEquivalent: katakanaBo  
 romajiEquivalent: romajiBo  
 },  
 {  
 hiraganaEquivalent: hiraganaPa  
 katakanaEquivalent: katakanaPa  
 romajiEquivalent: romajiPa  
 },  
 {  
 hiraganaEquivalent: hiraganaPi  
 katakanaEquivalent: katakanaPi  
 romajiEquivalent: romajiPi  
 },  
 {  
 hiraganaEquivalent: hiraganaPu  
 katakanaEquivalent: katakanaPu  
 romajiEquivalent: romajiPu  
 },  
 {  
 hiraganaEquivalent: hiraganaPe  
 katakanaEquivalent: katakanaPe  
 romajiEquivalent: romajiPe  
 },  
 {  
 hiraganaEquivalent: hiraganaPo  
 katakanaEquivalent: katakanaPo  
 romajiEquivalent: romajiPo  
 },  
 {  
 hiraganaEquivalent: hiraganaKya  
 katakanaEquivalent: katakanaKya  
 romajiEquivalent: romajiKya  
 },  
 {  
 hiraganaEquivalent: hiraganaKyu  
 katakanaEquivalent: katakanaKyu  
 romajiEquivalent: romajiKyu  
 },  
 {  
 hiraganaEquivalent: hiraganaKyo  
 katakanaEquivalent: katakanaKyo  
 romajiEquivalent: romajiKyo  
 },  
 {  
 hiraganaEquivalent: hiraganaSha  
 katakanaEquivalent: katakanaSha  
 romajiEquivalent: romajiSha  
 },  
 {  
 hiraganaEquivalent: hiraganaShu  
 katakanaEquivalent: katakanaShu  
 romajiEquivalent: romajiShu  
 },  
 {  
 hiraganaEquivalent: hiraganaSho  
 katakanaEquivalent: katakanaSho  
 romajiEquivalent: romajiSho  
 },  
 {  
 hiraganaEquivalent: hiraganaCha  
 katakanaEquivalent: katakanaCha  
 romajiEquivalent: romajiCha  
 },  
 {  
 hiraganaEquivalent: hiraganaChu  
 katakanaEquivalent: katakanaChu  
 romajiEquivalent: romajiChu  
 },  
 {  
 hiraganaEquivalent: hiraganaCho  
 katakanaEquivalent: katakanaCho  
 romajiEquivalent: romajiCho  
 },  
 {  
 hiraganaEquivalent: hiraganaHya  
 katakanaEquivalent: katakanaHya  
 romajiEquivalent: romajiHya  
 },  
 {  
 hiraganaEquivalent: hiraganaHyu  
 katakanaEquivalent: katakanaHyu  
 romajiEquivalent: romajiHyu  
 },  
 {  
 hiraganaEquivalent: hiraganaHyo  
 katakanaEquivalent: katakanaHyo  
 romajiEquivalent: romajiHyo  
 },  
 {  
 hiraganaEquivalent: hiraganaNya  
 katakanaEquivalent: katakanaNya  
 romajiEquivalent: romajiNya  
 },  
 {  
 hiraganaEquivalent: hiraganaNyu  
 katakanaEquivalent: katakanaNyu  
 romajiEquivalent: romajiNyu  
 },  
 {  
 hiraganaEquivalent: hiraganaNyo  
 katakanaEquivalent: katakanaNyo  
 romajiEquivalent: romajiNyo  
 },  
 {  
 hiraganaEquivalent: hiraganaMya  
 katakanaEquivalent: katakanaMya  
 romajiEquivalent: romajiMya  
 },  
 {  
 hiraganaEquivalent: hiraganaMyu  
 katakanaEquivalent: katakanaMyu  
 romajiEquivalent: romajiMyu  
 },  
 {  
 hiraganaEquivalent: hiraganaMyo  
 katakanaEquivalent: katakanaMyo  
 romajiEquivalent: romajiMyo  
 },  
 {  
 hiraganaEquivalent: hiraganaRya  
 katakanaEquivalent: katakanaRya  
 romajiEquivalent: romajiRya  
 },  
 {  
 hiraganaEquivalent: hiraganaRyu  
 katakanaEquivalent: katakanaRyu  
 romajiEquivalent: romajiRyu  
 },  
 {  
 hiraganaEquivalent: hiraganaRyo  
 katakanaEquivalent: katakanaRyo  
 romajiEquivalent: romajiRyo  
 },  
 {  
 hiraganaEquivalent: hiraganaGya  
 katakanaEquivalent: katakanaGya  
 romajiEquivalent: romajiGya  
 },  
 {  
 hiraganaEquivalent: hiraganaGyu  
 katakanaEquivalent: katakanaGyu  
 romajiEquivalent: romajiGyu  
 },  
 {  
 hiraganaEquivalent: hiraganaGyo  
 katakanaEquivalent: katakanaGyo  
 romajiEquivalent: romajiGyo  
 },  
 {  
 hiraganaEquivalent: hiraganaJya  
 katakanaEquivalent: katakanaJya  
 romajiEquivalent: romajiJya  
 },  
 {  
 hiraganaEquivalent: hiraganaJyu  
 katakanaEquivalent: katakanaJyu  
 romajiEquivalent: romajiJyu  
 },  
 {  
 hiraganaEquivalent: hiraganaJyo  
 katakanaEquivalent: katakanaJyo  
 romajiEquivalent: romajiJyo  
 },  
 {  
 hiraganaEquivalent: hiraganaDya  
 katakanaEquivalent: katakanaDya  
 romajiEquivalent: romajiDya  
 },  
 {  
 hiraganaEquivalent: hiraganaDyu  
 katakanaEquivalent: katakanaDyu  
 romajiEquivalent: romajiDyu  
 },  
 {  
 hiraganaEquivalent: hiraganaDyo  
 katakanaEquivalent: katakanaDyo  
 romajiEquivalent: romajiDyo  
 },  
 {  
 hiraganaEquivalent: hiraganaBya  
 katakanaEquivalent: katakanaBya  
 romajiEquivalent: romajiBya  
 },  
 {  
 hiraganaEquivalent: hiraganaByu  
 katakanaEquivalent: katakanaByu  
 romajiEquivalent: romajiByu  
 },  
 {  
 hiraganaEquivalent: hiraganaByo  
 katakanaEquivalent: katakanaByo  
 romajiEquivalent: romajiByo  
 },  
 {  
 hiraganaEquivalent: hiraganaPya  
 katakanaEquivalent: katakanaPya  
 romajiEquivalent: romajiPya  
 },  
 {  
 hiraganaEquivalent: hiraganaPyu  
 katakanaEquivalent: katakanaPyu  
 romajiEquivalent: romajiPyu  
 },  
 {  
 hiraganaEquivalent: hiraganaPyo  
 katakanaEquivalent: katakanaPyo  
 romajiEquivalent: romajiPyo  
 },  
]

[  
 {  
 kanjiNameID : kanjiHa  
 kanjiMeaning : leaf  
 kanjiMnemonic : The tree with the greatest (leaf)s laughs last **"HA" "HA"** hiraganaEquivalent : .hiraganaHa.  
 },  
 {  
 kanjiNameID : kanjiHana  
 kanjiMeaning : flower  
 kanjiMnemonic : Hannah is so pretty. No can stop planting (flower)s for **"Hannah"** hiraganaEquivalent : .hiraganaHa.hiraganaNa.  
 },  
 {  
 kanjiNameID : kanjiKi  
 kanjiMeaning : tree  
 kanjiMnemonic : To to go to Narnia. You need the **"key"** for the (tree)s door  
 hiraganaEquivalent : .hiraganaKi.  
  
 },  
 {  
 kanjiNameID : kanjiMori  
 kanjiMeaning : forest  
 kanjiMnemonic : To light a camp fire you will need **"more"** than one (forests) wood?  
 hiraganaEquivalent : .hiraganaMo.hiraganaRi.  
 },  
 {  
 kanjiNameID : kanjiMushi  
 kanjiMeaning : bug  
 kanjiMnemonic : How does squishing (insects) sound like **"mush"** hiraganaEquivalent : .hiraganaMu.hiraganaShi.  
 },  
 {  
 kanjiNameID : kanjiIshi  
 kanjiMeaning : stone  
 kanjiMnemonic : Breaking windows using a (stone) is surprisingly **"easy"** hiraganaEquivalent : .hiraganaI.hiraganaShi.  
 },  
 {  
 kanjiNameID : kanjiKaze  
 kanjiMeaning : wind  
 kanjiMnemonic : The word kamikaze was used in WWII. The **"kaze"** part means (wind).  
 hiraganaEquivalent: .hiraganaKa.hiraganaZe.  
  
 },  
 {  
 kanjiNameID : kanjiTsuchi  
 kanjiMeaning : soil  
 kanjiMnemonic : Minners who dig (soil) dream about the secret treasure **"two cheeses"** hiraganaEquivalent: .hiraganaTsu.hiraganaChi.  
  
 },  
 {  
 kanjiNameID : kanjiUchi  
 kanjiMeaning : inside  
 kanjiMnemonic : A resturant called **"Oh Cheese"** has just opened (inside) the mall  
 hiraganaEquivalent: .hiraganaU.hiraganaChi.  
  
 },  
 {  
 kanjiNameID : kanjiYama  
 kanjiMeaning : mountain  
 kanjiMnemonic : The **"yams"** found at the top of the (mountain) are very healthy  
 hiraganaEquivalent: .hiraganaYa.hiraganaMa.  
  
 },  
 {  
 kanjiNameID : kanjiUmi  
 kanjiMeaning : ocean  
 kanjiMnemonic : She sells g**"ummy me"**at by the (sea)shore  
 hiraganaEquivalent: .hiraganaU.hiraganaMi.  
  
 },  
 {  
 kanjiNameID : kanjiSakana  
 kanjiMeaning : fish  
 kanjiMnemonic : Fifa has organised a match between **"soccer nun"**s and (fish) market workers  
 hiraganaEquivalent: .hiraganaSa.hiraganaKa.hiraganaN.  
  
 },  
 {  
 kanjiNameID : kanjiMizu  
 kanjiMeaning : water  
 kanjiMnemonic : Have uou heard about the (water) misstress **"miss zu"** hiraganaEquivalent: .hiraganaMi.hiraganaZu.  
  
 },  
 {  
 kanjiNameID : kanjiKai  
 kanjiMeaning : shell  
 kanjiMnemonic : A **"ki"**te with the shape of a (shell) can fly the highest  
 hiraganaEquivalent: .hiraganaKa.hiraganaI.  
 },  
 {  
 kanjiNameID : kanjiIke  
 kanjiMeaning : pond  
 kanjiMnemonic : While having lunch near a (pond). A frog jumps on your food. You shriek **"EEEK"**.  
 hiraganaEquivalent: .hiraganaI.hiraganaKe.  
 },  
 {  
 kanjiNameID : kanjiYuki  
 kanjiMeaning : snow  
 kanjiMnemonic : The air is so polluted. The (snow) color is **"yucky"** blackish-green  
 hiraganaEquivalent: .hiraganaYu.hiraganaKi.  
 },  
 {  
 kanjiNameID : kanjiKoori  
 kanjiMeaning : ice  
 kanjiMnemonic : Cory from **"Cory"** in the house likes (ice) cream  
 hiraganaEquivalent: .hiraganaKo.hiraganaO.hiraganaRi.  
 },  
 {  
 kanjiNameID : kanjiKawa  
 kanjiMeaning : forest  
 kanjiMnemonic : Dont be **"a coward"** and jump to the (lake)  
 hiraganaEquivalent: .hiraganaKa.hiraganaWa.  
 },  
 {  
 kanjiNameID : kanjiHayashi  
 kanjiMeaning : forest  
 kanjiMnemonic : Hyundai new car can be driven in (forests) and its called Hyundai For She or **"hyushe"** hiraganaEquivalent: .hiraganaHa.hiraganaYa.hiraganaShi.  
 },  
 {  
 kanjiNameID : kanjiFuyu  
 kanjiMeaning : winter  
 kanjiMnemonic : Only **"few"** people can survive the (winter)  
 hiraganaEquivalent: .hiraganaFu.hiraganaYu.  
 },  
 {  
 kanjiNameID : kanjiTo  
 kanjiMeaning : door  
 kanjiMnemonic : Your friend accidentally closed the (door) on your **"toe"** hiraganaEquivalent: .hiraganaTo.  
 },  
 {  
 kanjiNameID : kanjiMon  
 kanjiMeaning : gate  
 kanjiMnemonic : Its said that **"mon"**sters live behind the (gate)  
 hiraganaEquivalent: .hiraganaMo.hiraganaN.  
 },  
 {  
 kanjiNameID : kanjiTera  
 kanjiMeaning : temple  
 kanjiMnemonic : The (temple) of friendship survived the **"terror"** attack  
 hiraganaEquivalent: .hiraganaTe.hiraganaRa.  
 },  
 {  
 kanjiNameID : kanjiAna  
 kanjiMeaning : hole  
 kanjiMnemonic : I had **"a na"**sty nightmare where I kept falling in a (hole)  
 hiraganaEquivalent: .hiraganaA.hiraganaNa.  
 },  
 {  
 kanjiNameID : kanjiSoto  
 kanjiMeaning : outside  
 kanjiMnemonic : Its winter. **"So to"** go (outside) you need a lot of jackets.  
 hiraganaEquivalent: .hiraganaSo.hiraganaTo.  
 },  
 {  
 kanjiNameID : kanjiKa  
 kanjiMeaning : fire  
 kanjiMnemonic : Camp (fire) can generate alot of **"he"**at  
 hiraganaEquivalent: .hiraganaHi.  
 },  
 {  
 kanjiNameID : kanjiHi  
 kanjiMeaning : sun  
 kanjiMnemonic : The (sun) can generate half as much **"he"**at as camp fire  
 hiraganaEquivalent: .hiraganaHi.  
 },  
 {  
 kanjiNameID : kanjiChi  
 kanjiMeaning : blood  
 kanjiMnemonic : (Blood) clotting can be improved with **"che"**ese  
 hiraganaEquivalent: .hiraganaChi.  
 },  
 {  
 kanjiNameID : kanjiAka  
 kanjiMeaning : red  
 kanjiMnemonic : My friends dont belive that I saw **"a ca"**t with (red) fur  
 hiraganaEquivalent: .hiraganaA.hiraganaKa.  
 },  
 {  
 kanjiNameID : kanjiHikari  
 kanjiMeaning : light  
 kanjiMnemonic : Only **"he carri"**es luggages faster than the speed of (light)  
 hiraganaEquivalent: .hiraganaHi.hiraganaKa.hiraganaRi.  
 },  
 {  
 kanjiNameID : kanjiAka  
 kanjiMeaning : temp  
 kanjiMnemonic : used for padding  
 hiraganaEquivalent: .hiraganaA.  
 },  
]

## Android\_Code

**package** com.mygdx.game;  
  
**import** android.os.Bundle;  
  
**import** com.badlogic.gdx.Application;  
**import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.backends.android.AndroidApplication;  
**import** com.badlogic.gdx.backends.android.AndroidApplicationConfiguration;  
**import** com.mygdx.game.screens.MainGameScreen;  
  
  
**public class** AndroidLauncher **extends** AndroidApplication {  
  
 AndroidApplicationConfiguration **config**;  
  
 @Override  
 **protected void** onCreate (Bundle savedInstanceState) {  
 **super**.onCreate(savedInstanceState);  
 **config** = **new** AndroidApplicationConfiguration();  
  
 initialize(**new** FinalKanjiQuest(), **config**);  
  
 Gdx.*app*.setLogLevel(Application.***LOG\_DEBUG***);  
 }  
  
 @Override  
 **protected void** onStart() {  
 **super**.onStart();  
 }  
  
 @Override  
 **protected void** onResume() {  
 **super**.onResume();  
 initialize(**new** FinalKanjiQuest(), **config**);  
 }  
  
 @Override  
 **protected void** onPause() {  
 **super**.onPause();  
 }  
  
 @Override  
 **protected void** onStop() {  
 **super**.onStop();  
 }  
  
 @Override  
 **protected void** onDestroy() {  
 **super**.onDestroy();  
 }  
}

**package** com.mygdx.game.components;  
  
**import** com.badlogic.gdx.Gdx;  
**import** com.badlogic.gdx.InputProcessor;  
**import** com.badlogic.gdx.math.Vector3;  
**import** com.mygdx.game.tools.Entity;  
**import** com.mygdx.game.screens.MainGameScreen;  
  
**public class** PlayerInputComponent **extends** InputComponent **implements** InputProcessor {  
  
 **private final static** String ***TAG*** = PlayerInputComponent.**class**.getSimpleName();  
 **private** Vector3 **lastMouseCoordinates**;  
 **boolean doNothing**;  
 **int screenWidth**;  
 **int screenHeight**;  
 **int touchDownX**;  
 **int touchDownY**;  
  
 **public** PlayerInputComponent(){  
 **this**.**lastMouseCoordinates** = **new** Vector3();  
  
 **screenWidth** = Gdx.*graphics*.getWidth();  
 **screenHeight** = Gdx.*graphics*.getHeight();  
 **touchDownX** = 0;  
 **touchDownY** = 0;  
 **doNothing** = **false**;  
  
 }  
  
 @Override  
 **public void** dispose() {  
 Gdx.*input*.setInputProcessor(**null**);  
 }  
  
 @Override  
 **public void** receiveMessage(String message) {  
 String[] string = message.split(***MESSAGE\_TOKEN***);  
  
 **if**( string.**length** == 0 ) **return**;  
  
 *//Specifically for messages with 1 object payload* **if**( string.**length** == 2 ) {  
 **if** (string[0].equalsIgnoreCase(MESSAGE.***CURRENT\_DIRECTION***.toString())) {  
 **currentDirection** = **json**.fromJson(Entity.Direction.**class**, string[1]);  
 }  
 }  
 }  
  
 @Override  
 **public void** update(Entity entity, **float** delta){  
  
 *//Keyboard input* **if**(*keys*.get(Keys.***PAUSE***)) {  
 MainGameScreen.*setGameState*(MainGameScreen.GameState.***PAUSED***);  
 pauseReleased();  
 }  
 **else if**( *keys*.get(Keys.***LEFT***)){  
 entity.sendMessage(MESSAGE.***CURRENT\_STATE***, **json**.toJson(Entity.State.***WALKING***));  
 entity.sendMessage(MESSAGE.***CURRENT\_DIRECTION***, **json**.toJson(Entity.Direction.***LEFT***));  
 }**else if**( *keys*.get(Keys.***RIGHT***)){  
 entity.sendMessage(MESSAGE.***CURRENT\_STATE***, **json**.toJson(Entity.State.***WALKING***));  
 entity.sendMessage(MESSAGE.***CURRENT\_DIRECTION***, **json**.toJson(Entity.Direction.***RIGHT***));  
 }**else if**( *keys*.get(Keys.***UP***)){  
 entity.sendMessage(MESSAGE.***CURRENT\_STATE***, **json**.toJson(Entity.State.***WALKING***));  
 entity.sendMessage(MESSAGE.***CURRENT\_DIRECTION***, **json**.toJson(Entity.Direction.***UP***));  
 }**else if**(*keys*.get(Keys.***DOWN***)){  
 entity.sendMessage(MESSAGE.***CURRENT\_STATE***, **json**.toJson(Entity.State.***WALKING***));  
 entity.sendMessage(MESSAGE.***CURRENT\_DIRECTION***, **json**.toJson(Entity.Direction.***DOWN***));  
 } **else if**( *keys*.get(Keys.***UP\_LEFT***)){  
 entity.sendMessage(MESSAGE.***CURRENT\_STATE***, **json**.toJson(Entity.State.***WALKING***));  
 entity.sendMessage(MESSAGE.***CURRENT\_DIRECTION***, **json**.toJson(Entity.Direction.***UP\_LEFT***));  
 }**else if**( *keys*.get(Keys.***UP\_RIGHT***)){  
 entity.sendMessage(MESSAGE.***CURRENT\_STATE***, **json**.toJson(Entity.State.***WALKING***));  
 entity.sendMessage(MESSAGE.***CURRENT\_DIRECTION***, **json**.toJson(Entity.Direction.***UP\_RIGHT***));  
 } **else if**(*keys*.get(Keys.***DOWN\_LEFT***)){  
 entity.sendMessage(MESSAGE.***CURRENT\_STATE***, **json**.toJson(Entity.State.***WALKING***));  
 entity.sendMessage(MESSAGE.***CURRENT\_DIRECTION***, **json**.toJson(Entity.Direction.***DOWN\_LEFT***));  
 } **else if**(*keys*.get(Keys.***DOWN\_RIGHT***)){  
 entity.sendMessage(MESSAGE.***CURRENT\_STATE***, **json**.toJson(Entity.State.***WALKING***));  
 entity.sendMessage(MESSAGE.***CURRENT\_DIRECTION***, **json**.toJson(Entity.Direction.***DOWN\_RIGHT***));  
 } **else if**(*keys*.get(Keys.***QUIT***)) {  
 quitReleased();  
 Gdx.*app*.exit();  
 } **else**{  
 entity.sendMessage(MESSAGE.***CURRENT\_STATE***, **json**.toJson(Entity.State.***IDLE***));  
 **if**( **currentDirection** == **null** ){  
 entity.sendMessage(MESSAGE.***CURRENT\_DIRECTION***, **json**.toJson(Entity.Direction.***DOWN***));  
 }  
 }  
  
 *//Mouse input* **if**( *mouseButtons*.get(Mouse.***SELECT***)) {  
 *//Gdx.app.debug(TAG, "Mouse LEFT click at : (" + lastMouseCoordinates.x + "," + lastMouseCoordinates.y + ")" );* entity.sendMessage(MESSAGE.***INIT\_SELECT\_ENTITY***, **json**.toJson(**lastMouseCoordinates**));  
 *mouseButtons*.put(Mouse.***SELECT***, **false**);  
 }  
 }  
  
 @Override  
 **public boolean** keyDown(**int** keycode) {  
 **return true**;  
 }  
  
 @Override  
 **public boolean** keyUp(**int** keycode) {  
 **return true**;  
 }  
  
 @Override  
 **public boolean** keyTyped(**char** character) {  
 **return false**;  
 }  
  
 @Override  
 **public boolean** touchDown(**int** screenX, **int** screenY, **int** pointer, **int** button) {  
   
 **touchDownX** = screenX;  
 **touchDownY** = screenY;  
 **this**.selectMouseButtonPressed(screenX, screenY);  
 **this**.setClickedMouseCoordinates(screenX, screenY);  
 *////Gdx.app.log(TAG, "touch down : (x == " + touchDownX + ", y ==" + touchDownY + ")" );  
  
 //top left corner* **if**((screenX >= 0 && screenX <= **screenWidth**/3) &&  
 (screenY >= 0 && screenY <= **screenHeight**/3)){  
 **if** (*keys*.get(Keys.***UP***) || *keys*.get(Keys.***DOWN***) || *keys*.get(Keys.***RIGHT***) ||  
 *keys*.get(Keys.***LEFT***) || *keys*.get(Keys.***UP\_RIGHT***) ||*keys*.get(Keys.***DOWN\_RIGHT***) ||  
 *keys*.get(Keys.***DOWN\_LEFT***)) {  
 **doNothing** = **true**;  
 *////Gdx.app.log(TAG, "top left doNothing is " + doNothing);* }  
 **else** {  
 *//Gdx.app.log(TAG, "up left pressed ");  
 keys*.put(Keys.***UP\_LEFT***, **true**);  
 }  
 }  
  
 *//top center* **if**((screenX >= **screenWidth**/3 && screenX <= **screenWidth**/1.5) &&  
 (screenY >= 0 && screenY <= **screenHeight**/3)){  
 **if** (*keys*.get(Keys.***DOWN***) || *keys*.get(Keys.***RIGHT***) ||  
 *keys*.get(Keys.***LEFT***) || *keys*.get(Keys.***UP\_RIGHT***) || *keys*.get(Keys.***UP\_LEFT***) ||*keys*.get(Keys.***DOWN\_RIGHT***) ||  
 *keys*.get(Keys.***DOWN\_LEFT***)) {  
 **doNothing** = **true**;  
 *//Gdx.app.log(TAG, "up doNothing is " + doNothing);* }  
 **else** {  
 *//Gdx.app.log(TAG, "up pressed ");  
 keys*.put(Keys.***UP***, **true**);  
 }  
 }  
  
 *//top right corner* **if**((screenX >= **screenWidth**/1.5 && screenX <= **screenWidth**) &&  
 (screenY >= 0 && screenY <= **screenHeight**/3)){  
 **if** (*keys*.get(Keys.***UP***) || *keys*.get(Keys.***DOWN***) || *keys*.get(Keys.***RIGHT***) ||  
 *keys*.get(Keys.***LEFT***) || *keys*.get(Keys.***UP\_LEFT***) ||*keys*.get(Keys.***DOWN\_RIGHT***) ||  
 *keys*.get(Keys.***DOWN\_LEFT***)) {  
 **doNothing** = **true**;  
 *//Gdx.app.log(TAG, "top right doNothing is " + doNothing);* }  
 **else** {  
 *//Gdx.app.log(TAG, "up right pressed ");  
 keys*.put(Keys.***UP\_RIGHT***, **true**);  
 }  
 }  
  
 *//center's left* **if**((screenX >= 0 && screenX <= **screenWidth**/3) &&  
 (screenY >= **screenHeight**/3 && screenY <= **screenHeight**/1.5)){  
 **if** (*keys*.get(Keys.***UP***) || *keys*.get(Keys.***DOWN***) || *keys*.get(Keys.***RIGHT***) ||  
 *keys*.get(Keys.***UP\_RIGHT***) || *keys*.get(Keys.***UP\_LEFT***) ||*keys*.get(Keys.***DOWN\_RIGHT***) ||  
 *keys*.get(Keys.***DOWN\_LEFT***)) {  
 **doNothing** = **true**;  
 *//Gdx.app.log(TAG, "left doNothing is " + doNothing);* }  
 **else** {  
 *//Gdx.app.log(TAG, "left pressed ");  
 keys*.put(Keys.***LEFT***, **true**);  
 }  
 }  
 *//center* **if**((screenX >= **screenWidth**/3 && screenX <= **screenWidth**/1.5) &&  
 (screenY >= **screenHeight**/3 && screenY <= **screenHeight**/1.5)){  
 *//do nothing for now* **doNothing** = **true**;  
 *//Gdx.app.log(TAG, "center doNothing is " + doNothing);* }  
  
 *//center's right pressed* **if**((screenX >= **screenWidth**/1.5 && screenX <= **screenWidth**) &&  
 (screenY >= **screenHeight**/3 && screenY <= **screenHeight**/1.5)){  
 **if** (*keys*.get(Keys.***UP***) || *keys*.get(Keys.***DOWN***) || *keys*.get(Keys.***LEFT***) ||  
 *keys*.get(Keys.***UP\_RIGHT***) || *keys*.get(Keys.***UP\_LEFT***) ||*keys*.get(Keys.***DOWN\_RIGHT***) ||  
 *keys*.get(Keys.***DOWN\_LEFT***)) {  
 **doNothing** = **true**;  
 *//Gdx.app.log(TAG, "right doNothing is " + doNothing);* }  
 **else** {  
 *//Gdx.app.log(TAG, "right pressed ");  
 keys*.put(Keys.***RIGHT***, **true**);  
 }  
 }  
  
 *//bottom left corner* **if**((screenX >= 0 && screenX <= **screenWidth**/3) &&  
 (screenY >= **screenHeight**/1.5 && screenY <= **screenHeight**)){  
 **if** (*keys*.get(Keys.***UP***) || *keys*.get(Keys.***DOWN***) || *keys*.get(Keys.***RIGHT***) ||  
 *keys*.get(Keys.***LEFT***) || *keys*.get(Keys.***UP\_RIGHT***) || *keys*.get(Keys.***UP\_LEFT***) ||  
 *keys*.get(Keys.***DOWN\_RIGHT***)) {  
 **doNothing** = **true**;  
 *//Gdx.app.log(TAG, "down left doNothing is " + doNothing);* }  
 **else** {  
 *//Gdx.app.log(TAG, "down left pressed ");  
 keys*.put(Keys.***DOWN\_LEFT***, **true**);  
 }  
 }  
  
 *//bottom center* **if**((screenX >= **screenWidth**/3 && screenX <= **screenWidth**/1.5) &&  
 (screenY >= **screenHeight**/1.5 && screenY <= **screenHeight**)){  
 **if** (*keys*.get(Keys.***UP***) || *keys*.get(Keys.***RIGHT***) || *keys*.get(Keys.***LEFT***) ||  
 *keys*.get(Keys.***UP\_RIGHT***) || *keys*.get(Keys.***UP\_LEFT***) ||*keys*.get(Keys.***DOWN\_RIGHT***) ||  
 *keys*.get(Keys.***DOWN\_LEFT***)) {  
 **doNothing** = **true**;  
 *//Gdx.app.log(TAG, "down doNothing is " + doNothing);* }  
 **else** {  
 *//Gdx.app.log(TAG, "down pressed ");  
 keys*.put(Keys.***DOWN***, **true**);;  
 }  
 }  
  
 *//bottom right corner* **if**((screenX >= **screenWidth**/1.5 && screenX <= **screenWidth**) &&  
 (screenY >= **screenHeight**/1.5 && screenY <= **screenHeight**)){  
 **if** (*keys*.get(Keys.***UP***) || *keys*.get(Keys.***DOWN***) || *keys*.get(Keys.***RIGHT***) ||  
 *keys*.get(Keys.***LEFT***) || *keys*.get(Keys.***UP\_RIGHT***) || *keys*.get(Keys.***UP\_LEFT***) ||  
 *keys*.get(Keys.***DOWN\_LEFT***)) {  
 **doNothing** = **true**;  
 *//Gdx.app.log(TAG, "down right doNothing is " + doNothing);* }  
 **else** {  
 *//Gdx.app.log(TAG, "down right pressed ");  
 keys*.put(Keys.***DOWN\_RIGHT***, **true**);;  
 }  
 }  
  
 *//Gdx.app.log(TAG, "touch down returned" );* **return true**;  
 }  
  
 @Override  
 **public boolean** touchUp(**int** screenX, **int** screenY, **int** pointer, **int** button) {  
   
 **this**.selectMouseButtonReleased(screenX, screenY);  
  
 *////Gdx.app.log(TAG, "touch up : ( x == " + screenX + ", y ==" + screenY + ")" );  
 //Gdx.app.log(TAG, "touch up inaction, doNothing: " + doNothing);* **if**(**doNothing**){  
 *////Gdx.app.log(TAG, "doing nothing" );* }  
 **else** {  
 *////Gdx.app.log(TAG, "hide() called" );  
 hide*();  
 }  
  
 **doNothing** = **false**;  
  
 **return true**;  
 }  
  
 @Override  
 **public boolean** touchDragged(**int** screenX, **int** screenY, **int** pointer) {  
 **return false**;  
 }  
  
 @Override  
 **public boolean** mouseMoved(**int** screenX, **int** screenY) {  
 **return false**;  
 }  
  
 @Override  
 **public boolean** scrolled(**int** amount) {  
 **return false**;  
 }  
  
 **public void** quitPressed(){  
 *keys*.put(Keys.***QUIT***, **true**);  
 }  
  
 **public void** pausePressed() {  
 *keys*.put(Keys.***PAUSE***, **true**);  
 }  
   
 **public void** quitReleased(){  
 *keys*.put(Keys.***QUIT***, **false**);  
 }  
  
 **public void** pauseReleased() { *keys*.put(Keys.***PAUSE***, **false**);}  
  
 **public void** setClickedMouseCoordinates(**int** x,**int** y){  
 **lastMouseCoordinates**.set(x, y, 0);  
 }  
  
 **public void** selectMouseButtonPressed(**int** x, **int** y){  
 *mouseButtons*.put(Mouse.***SELECT***, **true**);  
 }  
   
 **public void** selectMouseButtonReleased(**int** x, **int** y){  
 *mouseButtons*.put(Mouse.***SELECT***, **false**);  
 }  
  
 **public static void** hide(){  
 *//Gdx.app.log(TAG, "hide() inaction" );  
 keys*.put(Keys.***LEFT***, **false**);  
 *keys*.put(Keys.***RIGHT***, **false**);  
 *keys*.put(Keys.***UP***, **false**);  
 *keys*.put(Keys.***DOWN***, **false**);  
 *keys*.put(Keys.***UP\_RIGHT***, **false**);  
 *keys*.put(Keys.***UP\_LEFT***, **false**);  
 *keys*.put(Keys.***DOWN\_RIGHT***, **false**);  
 *keys*.put(Keys.***DOWN\_LEFT***, **false**);  
 }  
  
}