MW-A SORE

STMIK MIKROSKIL

12.111.0090 – FRENDY

12.111.0120 – SHERLY F

12.111.0995 – HARTONO P

12.111.0324 – STEVEN H

12.111.0146 – KATARINA Y V Z

|  |  |
| --- | --- |
| http://prod-images.exhibit-e.com/www_houkgallery_com/COWBOY_272_19990.jpg  1 2 3 Dor!  Pemrograman Mobile Lanjutan – Tugas Besar | ANGGOTA |

# Deskripsi Singkat

Aplikasi untuk game “1..2..3.. Dor!” dibuat dengan menggunakan framework libGDX oleh badlogicgames. Mengembangkan game dengan framework tersebut memudahkan dalam segi pembagian logika pengerjaan game. Prinsip pengerjaannya dibagi dengan beberapa konsep utama yakni, Screen, Stage, dan Actor. Screen digunakan untuk melakukan pemindahan dari satu tampilan ke tampilan lainnya. Stage digunakan untuk menampung Actor dan menangani aksi pada stage. Pada pembuatan game ini, class dengan tujuan yang sama ditempatkan pada folder yang sama.

# Function & Procedure

## Class dan Fungsi – fungsi dasar yang penting

### AndroidLauncher.java

File ini memuat class androidlauncher yang meng-extend AndroidApplication. Tujuan dari class ini cukup sederhana, yakni menampilkan launcher(bila ada) dan menjalankan program utama. Di dalam onCreate() class ini menginisialisasi konfigurasi aplikasi dan memanggil program utama(CowboyDor).

public class AndroidLauncher extends AndroidApplication {  
 @Override  
 protected void onCreate (Bundle savedInstanceState) {  
 super.onCreate(savedInstanceState);  
 AndroidApplicationConfiguration config = new AndroidApplicationConfiguration();  
 initialize(new CowboyDor(), config);  
 }  
}

### CowboyDor.java

Memuat class CowboyDor yang meng-extend Game. Di dalam class tersebut dilakukan fungsi create untuk inisialisasi ScreenManager dan menampilkan Screen utama yang diinginkan, yakni Main Menu.

public class CowboyDor extends Game {  
  
 public void create() {  
 ScreenManager.*getInstance*().initialize(this);  
 ScreenManager.*getInstance*().showScreen( ScreenEnum.*MAIN\_MENU* );  
 }  
  
 public void render() {  
 super.render(); //important!  
 }  
  
 public void dispose() {  
 }  
}

### Cowboy.java

Memuat class Cowboy. Class cowboy merupakan object yang dapat digunakan sebagai actor dalam Stage.

public class Cowboy {  
  
 public enum State {  
 *IDLE*, *SHOOT*, *WIN*, *LOSE*, *HIDE* }  
  
 static final float *SIZE* = 1f; // half a unit  
 int cowboyId;  
 int score=0;  
 int score\_limit = 0;  
 float speed;  
 float topSpeed = 0;  
 Vector2 position = new Vector2();  
 Rectangle bounds = new Rectangle();  
 State state = State.*IDLE*;  
 boolean flagHit = false;  
 boolean flagShot = false;  
 boolean flagReady = false;  
  
 public State getState(){  
 return state;  
 }  
 public void setState(State state){  
 this.state = state;  
 }  
  
 public int getCowboyId(){  
 return cowboyId;  
 }  
  
 public Rectangle getBounds(){  
 return this.bounds;  
 }  
  
 public Vector2 getPosition(){  
 return this.position;  
 }  
  
 public void setScore\_limit(int limit){  
 this.score\_limit = limit;  
 }  
 public int getScore\_limit(){  
 return this.score\_limit;  
 }  
  
 public void setTopSpeed(float speed){  
 this.topSpeed = speed;  
 }  
 public float getTopSpeed(){  
 return this.topSpeed;  
 }  
  
 public void setSpeed(float speed){  
 this.speed = speed;  
 }  
 public float getSpeed(){  
 return this.speed;  
 }  
  
 public int getScore(){  
 return this.score;  
 }  
 public void setScore(int score){  
 this.score = score;  
 }  
 public void addScore(){  
 this.score++;  
 }  
 public void resetScore(){ this.score = 0;}  
  
 public boolean isFlagHit(){  
 return this.flagHit;  
 }  
 public void setFlagHit(boolean flag){  
 this.flagHit = flag;  
 }  
  
 public boolean isFlagShot(){  
 return this.flagShot;  
 }  
 public void setFlagShot(boolean flag){  
 this.flagShot = flag;  
 }  
  
 public boolean isFlagReady() {  
 return this.flagReady;  
 }  
 public void setFlagReady(boolean flag){  
 this.flagReady = flag;  
 }  
  
 public float getSize(){  
 return this.*SIZE*;  
 }  
  
 public Cowboy(int id, Vector2 pos){  
 this.cowboyId = id;  
 this.position = pos;  
 this.bounds.height = 1f;  
 this.bounds.width = 1f;  
 }  
}

### GameMode.java

Memuat class GameMode untuk menentukan mode game yang diinginkan. Masing – masing mode game disertakan dengan settingan environment tersendiri. Seperti jumlah ronde yang diinginkan, tipe game, limit score, dan mencatat ronde.

public class GameMode {  
  
 public enum GameType {  
 *CLASSIC*, *FIRST\_TO\_1*, *FIRST\_TO\_3*, *FIRST\_TO\_5* }  
  
 private int round=0;  
 private int round\_limit=3;  
 private int score\_limit = 0;  
 private boolean flagUseRound = true;  
  
 GameType type = GameType.*CLASSIC*;  
  
 public GameMode(GameType mode){  
 if(mode.equals(GameType.*CLASSIC*)){  
 this.type = GameType.*CLASSIC*;  
 setRound(1);  
 }else if(mode.equals(GameType.*FIRST\_TO\_1*)) {  
 this.type = GameType.*FIRST\_TO\_1*;  
 this.flagUseRound = false;  
 this.score\_limit = 1;  
 }else if(mode.equals(GameType.*FIRST\_TO\_3*)){  
 this.type = GameType.*FIRST\_TO\_3*;  
 this.flagUseRound = false;  
 this.score\_limit = 3;  
 }else if(mode.equals(GameType.*FIRST\_TO\_5*)) {  
 this.type = GameType.*FIRST\_TO\_5*;  
 this.flagUseRound = false;  
 this.score\_limit = 5;  
 }else{  
 this.type = GameType.*CLASSIC*;  
 setRound(1);  
 }  
 }  
  
 public int getScore\_limit(){  
 return this.score\_limit;  
 }  
  
 public void setRound(int round){  
 this.round = round;  
 }  
 public int getRound(){  
 return this.round;  
 }  
 public boolean nextRound(){  
 if((this.round + 1) <= this.round\_limit){  
 this.round = this.round+1;  
 return true;  
 }else{  
 return false;  
 }  
 }  
  
 public GameType getGameType(){  
 return this.type;  
 }  
  
}

### World.java

Memuat class world. Class ini digunakan ketika melakukan inisialisasi terhadap tampilan permainan dimulai. Class ini akan memasukkan masing – masing cowboy ke posisinya.

public class World {  
  
 Array<Cowboy> cowboys = new Array<Cowboy>();  
  
 public Array<Cowboy> getCowboys(){  
 return cowboys;  
 }  
  
 public World(){  
 createWorld();  
 }  
  
 private void createWorld(){  
 cowboys.add(new Cowboy(1, new Vector2(2,3)));  
 cowboys.add(new Cowboy(2, new Vector2(7,3)));  
 }  
  
}

## Utility

Game ini terdiri dari 3 buah utility sederhana yang digunakan untuk manajemen Screen dan menambahkan komponen dengan mudah. 3 utility tersebut diantaranya :

1. Screen Enum

Screen enum digunakan untuk menginisialisasi dan mempermudah pemanggilan Screen. Tugas dari screen enum cukup sederhana yakni menampung dan menginisialisasi setiap Screen yang ada pada game untuk mempermudah pemanggilan.

public enum ScreenEnum {  
  
 *MAIN\_MENU* {  
 public AbstractScreen getScreen(Object... params) {  
 return new MainMenuScreen();  
 }  
 },  
  
 *LEVEL\_SELECT* {  
 public AbstractScreen getScreen(Object... params) {  
 return new LevelSelectScreen();  
 }  
 },  
  
 *GAME* {  
 public AbstractScreen getScreen(Object... params) {  
 return new GameScreen((GameMode.GameType) params[0], false);  
 }  
 };  
  
 public abstract AbstractScreen getScreen(Object... params);  
}

1. Screen Manager

Screen Manager digunakan untuk manajemen Screen seperti menampilkan dan menghapus Screen beserta komponen – komponen yang ada di dalamnya. Fungsi Show Screen di dalam class tersebut akan menampilkan Screen yang dipilih dengan menerima parameter Screen enum dan parameter data lain.

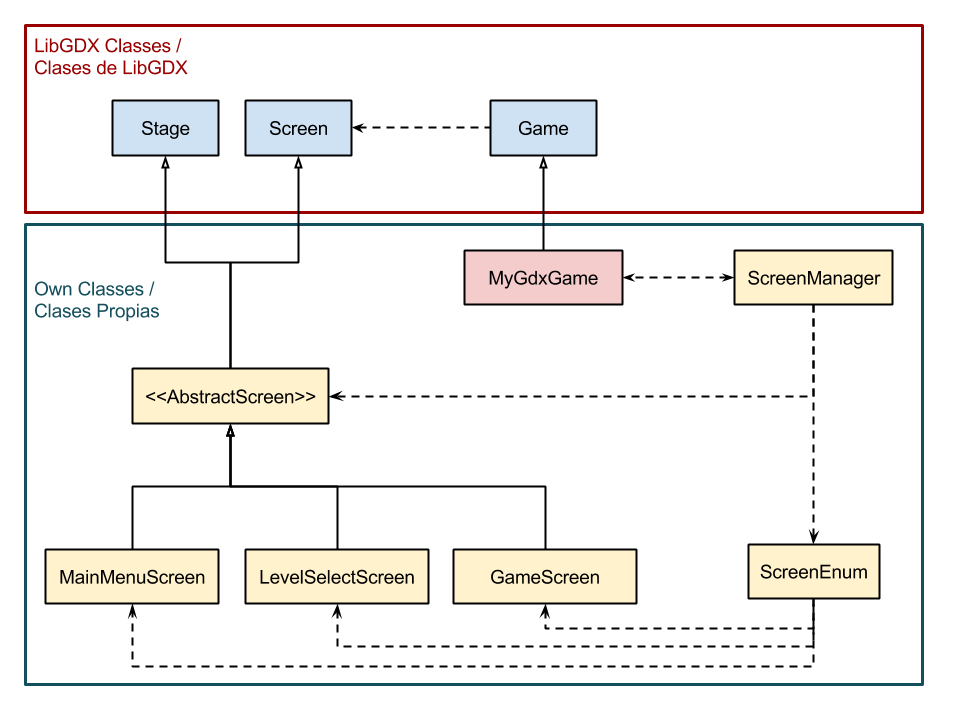
public class ScreenManager {  
  
 // Singleton: unique instance  
 private static ScreenManager *instance*;  
  
 // Reference to game  
 private Game game;  
  
 // Singleton: private constructor  
 private ScreenManager() {  
 super();  
 }  
  
 // Singleton: retrieve instance  
 public static ScreenManager getInstance() {  
 if (*instance* == null) {  
 *instance* = new ScreenManager();  
 }  
 return *instance*;  
 }  
  
 // Initialization with the game class  
 public void initialize(Game game) {  
 this.game = game;  
 }  
  
 // Show in the game the screen which enum type is received  
 public void showScreen(ScreenEnum screenEnum, Object... params) {  
  
 // Get current screen to dispose it  
 Screen currentScreen = game.getScreen();  
  
 // Show new screen  
 AbstractScreen newScreen = screenEnum.getScreen(params);  
 newScreen.buildStage();  
 game.setScreen(newScreen);  
  
 // Dispose previous screen  
 if (currentScreen != null) {  
 currentScreen.dispose();  
 }  
 }  
}

1. UIFactory

Fungsi UIFactory cukup sederhana yakni, dapat digunakan untuk membuat button dan juga dapat digunakan untuk menginisialisasi Listener pada button yang sudah dibuat.

public class UIFactory {  
  
 public static ImageButton createButton(Texture texture) {  
 return  
 new ImageButton(  
 new TextureRegionDrawable(  
 new TextureRegion(texture) ) );  
 }  
  
 public static InputListener createListener(final ScreenEnum dstScreen, final Object... params) {  
 return  
 new InputListener() {  
 @Override  
 public boolean touchDown(InputEvent event, float x,  
 float y, int pointer, int button) {  
 ScreenManager.*getInstance*().showScreen(dstScreen, params);  
 return false;  
 }  
 };  
 }  
}

## Screen



### AbstractScreen

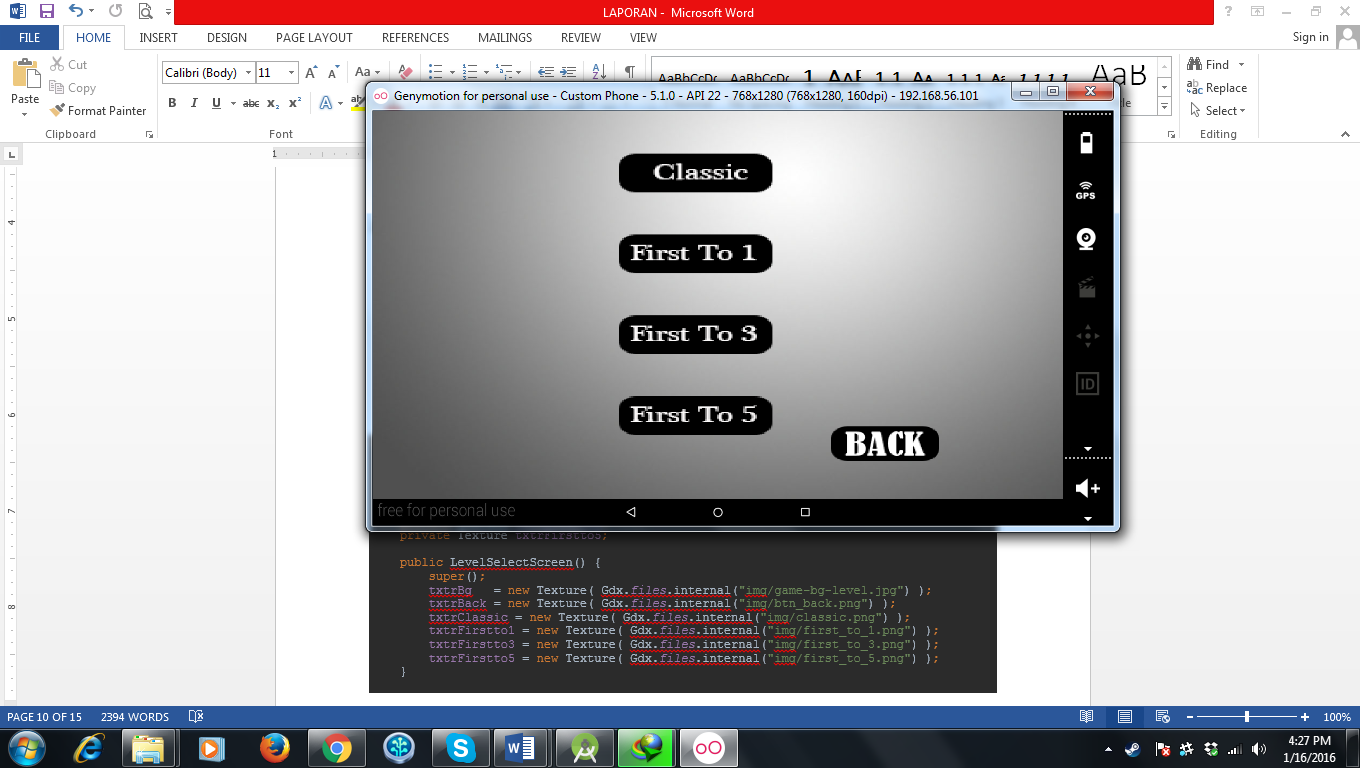
AbstractScreen digunakan sebagai class screen dasar untuk menampilkan screen lainnya. Screen lainnya akan menampilkan isi dengan ciri – ciri yang disamakan melalui AbstractScreen tanpa harus di definiskan ulang pada masing – masing screen. Screen pada dasarnya dapat juga digunakan untuk menentukan lebar dan panjang dari viewport aplikasi. Penentuan ini berguna untuk menyamakan titik perhitungan viewport pada ukuran layar yang berbeda.

public abstract class AbstractScreen extends Stage implements Screen {  
  
 protected AbstractScreen() {  
 super( new StretchViewport(320.0f, 240.0f, new OrthographicCamera()) );  
 }  
  
 // Subclasses must load actors in this method  
 public abstract void buildStage();  
  
 @Override  
 public void render(float delta) {  
 // Clear screen  
 Gdx.*gl*.glClearColor(0, 0, 0, 1);  
 Gdx.*gl*.glClear(GL20.*GL\_COLOR\_BUFFER\_BIT*);  
  
 // Calling to Stage methods  
 super.act(delta);  
 super.draw();  
 }  
  
 @Override  
 public void show() {  
 Gdx.*input*.setInputProcessor(this);  
 }  
  
 @Override  
 public void resize(int width, int height) {  
 getViewport().update(width, height);  
 }  
  
 @Override public void hide() {}  
 @Override public void pause() {}  
 @Override public void resume() {}  
}

fungsi buildStage() harus dipanggil untuk setiap Screen yang meng-extends AbstractScreen. Fungsi buildStage ini digunakan untuk membuat stage dan memasukkan actor ke dalam stage.

### LevelSelectScreen

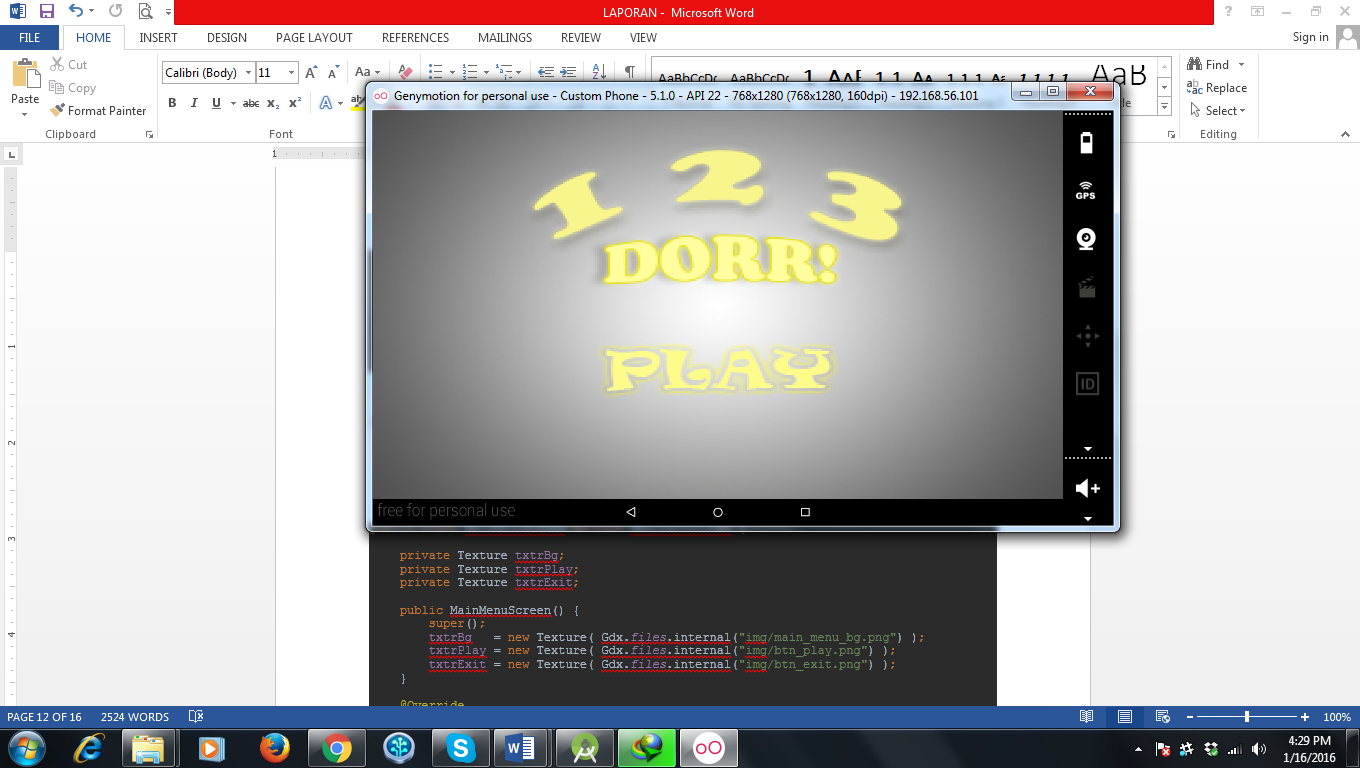
Tujuan dari Screen ini cukup sederhana. LevelSelectScreen meng-extends AbstractScreen dan memiliki ciri – ciri yang sama. LevelSelectScreen diwajibkan untuk meng-override buildStage dan kemudian dapat digunakan untuk meletakkan actor di dalamnya. Dalam screen ini, kami meletakkan enam jenis actor diantaranya background, tombol back, dan sisanya adalah tombol untuk mode game. Untuk berpindah halaman, setiap button diberikan sebuah listener dengan memberikan parameter – parameter yang sesuai. Contohnya seperti tombol back yang diberikan sebuah listener untuk dapat kembali ke MainMenuScreen.



public class LevelSelectScreen extends AbstractScreen {  
  
 private Texture txtrBg;  
 private Texture txtrBack;  
 private Texture txtrClassic;  
 private Texture txtrFirstto1;  
 private Texture txtrFirstto3;  
 private Texture txtrFirstto5;  
  
 public LevelSelectScreen() {  
 super();  
 txtrBg = new Texture( Gdx.*files*.internal("img/game-bg-level.jpg") );  
 txtrBack = new Texture( Gdx.*files*.internal("img/btn\_back.png") );  
 txtrClassic = new Texture( Gdx.*files*.internal("img/classic.png") );  
 txtrFirstto1 = new Texture( Gdx.*files*.internal("img/first\_to\_1.png") );  
 txtrFirstto3 = new Texture( Gdx.*files*.internal("img/first\_to\_3.png") );  
 txtrFirstto5 = new Texture( Gdx.*files*.internal("img/first\_to\_5.png") );  
 }  
  
 @Override  
 public void buildStage() {  
  
 // Adding actors  
 Image bg = new Image(txtrBg);  
 bg.setSize(480,360);  
 addActor(bg);  
  
 ImageButton btnBack = UIFactory.*createButton*(txtrBack);  
 btnBack.setPosition(260.f, 40.f, Align.*center*);  
 btnBack.setSize(50, 30);  
 addActor(btnBack);  
  
 //position is from bottom left ( x from left, y from bottom )  
 ImageButton btnClassic = UIFactory.*createButton*(txtrClassic);  
 btnClassic.setPosition(160, 200.f, Align.*center*);  
 btnClassic.setSize(80, 50);  
 addActor(btnClassic);  
  
 ImageButton btnFirstto1 = UIFactory.*createButton*(txtrFirstto1);  
 btnFirstto1.setPosition(160, 150.f, Align.*center*);  
 btnFirstto1.setSize(80, 50);  
 addActor(btnFirstto1);  
  
 ImageButton btnFirstto3 = UIFactory.*createButton*(txtrFirstto3);  
 btnFirstto3.setPosition(160, 100.f, Align.*center*);  
 btnFirstto3.setSize(80, 50);  
 addActor(btnFirstto3);  
  
 ImageButton btnFirstto5 = UIFactory.*createButton*(txtrFirstto5);  
 btnFirstto5.setPosition(160, 50.f, Align.*center*);  
 btnFirstto5.setSize(80, 50);  
 addActor(btnFirstto5);  
  
 btnBack.addListener(UIFactory.*createListener*(ScreenEnum.*MAIN\_MENU*));  
 btnClassic.addListener( UIFactory.*createListener*(ScreenEnum.*GAME*, GameMode.GameType.*CLASSIC*) );  
 btnFirstto1.addListener( UIFactory.*createListener*(ScreenEnum.*GAME*, GameMode.GameType.*FIRST\_TO\_1*) );  
 btnFirstto3.addListener( UIFactory.*createListener*(ScreenEnum.*GAME*, GameMode.GameType.*FIRST\_TO\_3*) );  
 btnFirstto5.addListener( UIFactory.*createListener*(ScreenEnum.*GAME*, GameMode.GameType.*FIRST\_TO\_5*) );  
 }  
  
 @Override  
 public void dispose() {  
 super.dispose();  
 txtrBg.dispose();  
 txtrBack.dispose();  
 txtrClassic.dispose();  
 txtrFirstto1.dispose();  
 txtrFirstto3.dispose();  
 txtrFirstto5.dispose();  
 }  
}

### MainMenuScreen

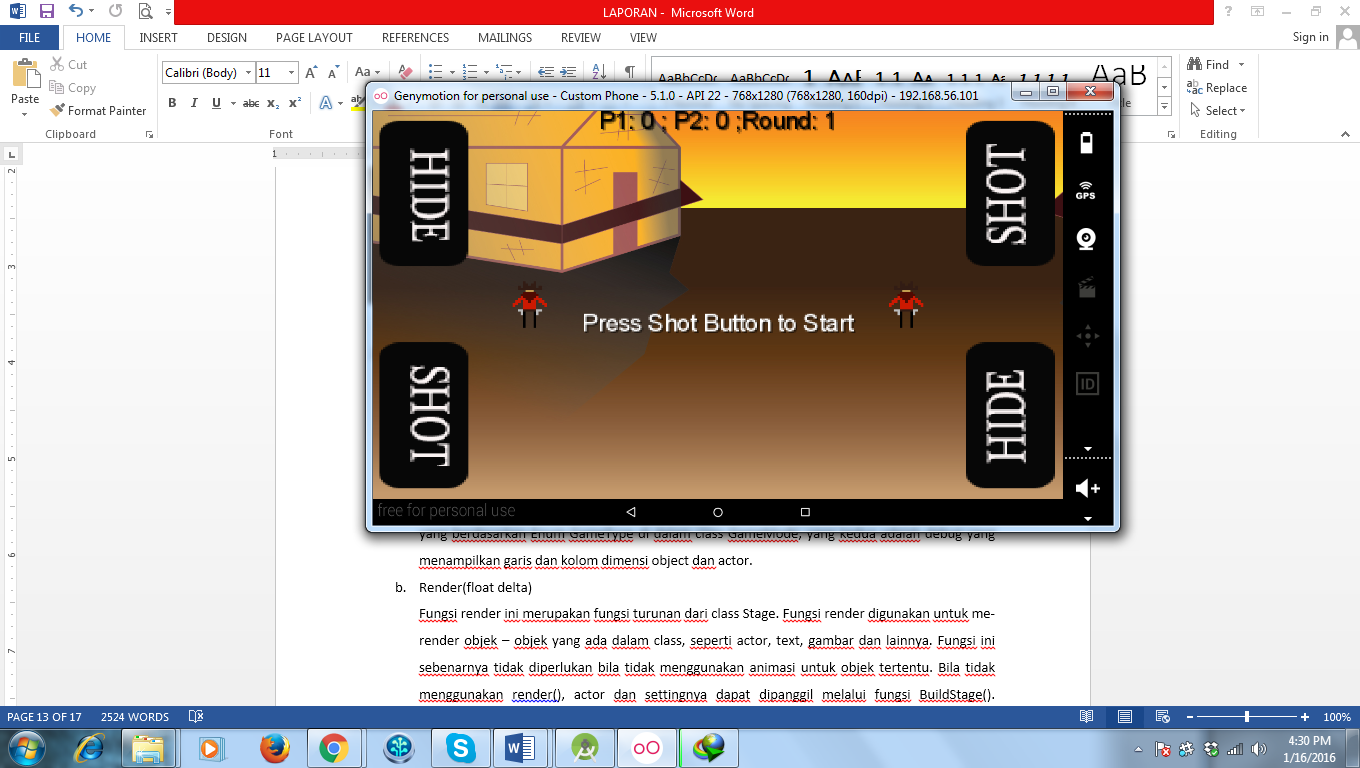
Hampir mirip dengan LevelSelectScreen, hanya saja MainMenuScreen hanya memuat 2 actor, yakni background dan tombol play. Tombol play diberikan listener yang dapat mengganti screen ke LevelSelectScreen.

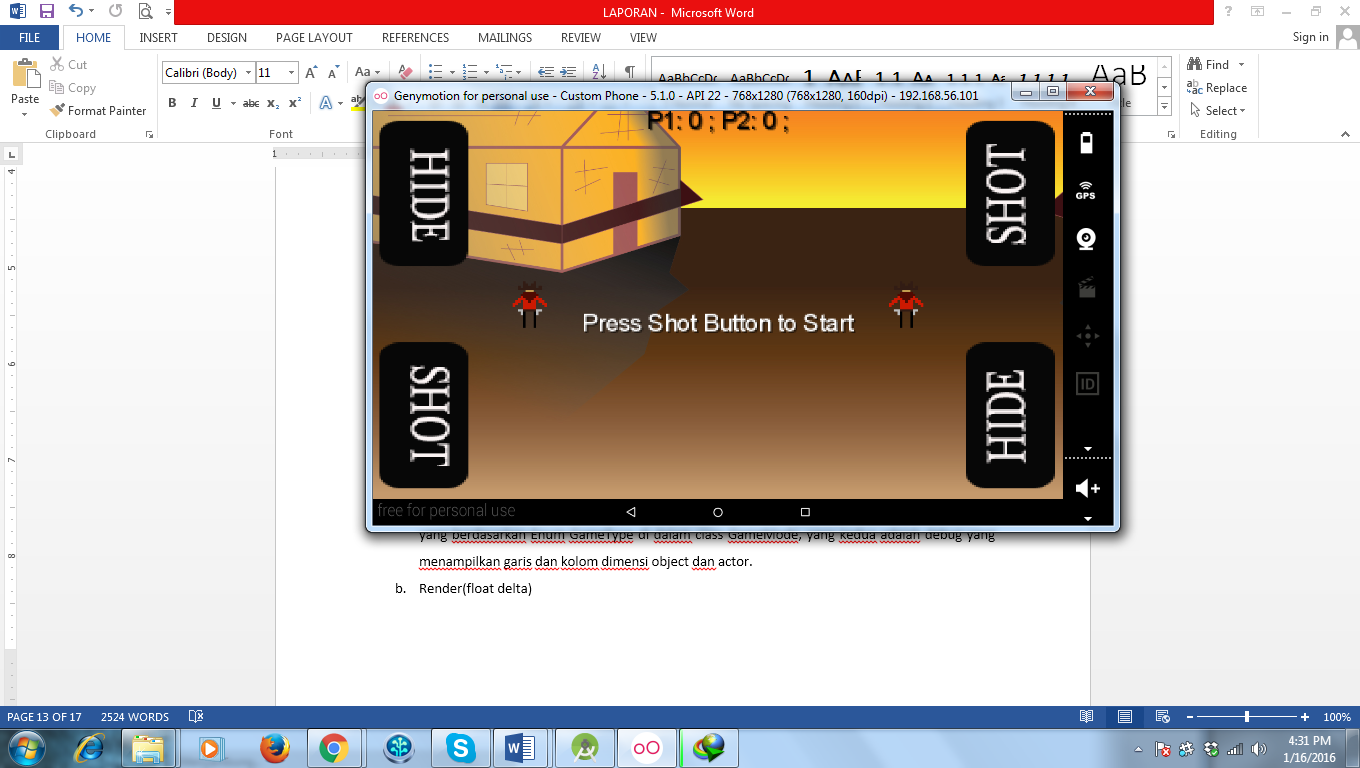


public class MainMenuScreen extends AbstractScreen {  
  
 private Texture txtrBg;  
 private Texture txtrPlay;  
 private Texture txtrExit;  
  
 public MainMenuScreen() {  
 super();  
 txtrBg = new Texture( Gdx.*files*.internal("img/main\_menu\_bg.png") );  
 txtrPlay = new Texture( Gdx.*files*.internal("img/btn\_play.png") );  
 txtrExit = new Texture( Gdx.*files*.internal("img/btn\_exit.png") );  
 }  
  
 @Override  
 public void buildStage() {  
  
 // Adding actors  
 Image bg = new Image(txtrBg);  
 addActor(bg);  
  
 ImageButton btnPlay = UIFactory.*createButton*(txtrPlay);  
 // btnPlay.setPosition(getWidth() / 2, 120.f, Align.center);  
 btnPlay.setPosition(getWidth() / 2, 80.f, Align.*center*);  
 addActor(btnPlay);  
  
 // Setting listeners  
 btnPlay.addListener( UIFactory.*createListener*(ScreenEnum.*LEVEL\_SELECT*) );  
 }  
  
 @Override  
 public void dispose() {  
 super.dispose();  
 txtrBg.dispose();  
 txtrPlay.dispose();  
 txtrExit.dispose();  
 }  
}

### GameScreen

Game Screen juga memiliki ciri – ciri yang serupa. GameScreen merupakan Screen yang digunakan untuk melakukan permainan berdasarkan mode – mode yang dipilih pada LevelSelectScreen.





Terdapat beberapa fungsi dan prosedur penting yang digunakan di dalam Class Game Screen diantaranya:

1. Constructor GameScreen(GameType GT, boolean debug)

Constructor GameScreen menerima dua jenis parameter yakni yang pertama adalah GameType yang berdasarkan Enum GameType di dalam class GameMode, yang kedua adalah debug yang menampilkan garis dan kolom dimensi object dan actor.

public GameScreen(GameMode.GameType GT, boolean debug) {  
 super();  
 this.world = new World();  
 this.gameMode = new GameMode(GT);  
 this.cam = new OrthographicCamera(*CAMERA\_WIDTH*, *CAMERA\_HEIGHT*);  
 this.cam.position.set(*CAMERA\_WIDTH* / 2f, *CAMERA\_HEIGHT* / 2f, 0);  
 this.cam.update();  
 this.debug = debug;  
 spriteBatch = new SpriteBatch();  
 scoreBoardBitmap = new BitmapFont();  
 counterBitmap = new BitmapFont();  
 cowboy1NotifBitmap = new BitmapFont();  
 cowboy2NotifBitmap = new BitmapFont();  
 scoreBoardText = "P1: 0 ; P2: 0 ;";  
 if(gameMode.getGameType().equals(GameMode.GameType.*CLASSIC*))  
 scoreBoardText+= "Round: 1";  
 counterText = "Touch to Start";  
 cowboy1Notif = "";  
 cowboy2Notif = "";  
  
 stateTime =0 ;  
 loadTextures();  
 loadSound();  
 loadGameMode();  
}

1. Render(float delta)

Fungsi render ini merupakan fungsi turunan dari class Stage. Fungsi render digunakan untuk me-render objek – objek yang ada dalam class, seperti actor, text, gambar dan lainnya. Fungsi ini sebenarnya tidak diperlukan bila tidak menggunakan animasi untuk objek tertentu. Bila tidak menggunakan render(), actor dan settingnya dapat dipanggil melalui fungsi BuildStage().   
Dalam game ini, kami menggunakan fungsi render untuk melakukan Count Down tembakan cowboy dan perhitungan speed, meletakkan background dan sprite, animasi cowboy berdasarkan state, dan refresh mode Game.

@Override  
public void render(float delta) {  
 Gdx.*gl*.glClearColor(0, 1, 0, 0.8f);  
 Gdx.*gl*.glClear(GL20.*GL\_COLOR\_BUFFER\_BIT*);  
  
 spriteBatch.begin();  
 drawBackground(spriteBatch);  
 drawCounter(spriteBatch);  
 drawCowboy();  
 drawGame(spriteBatch);  
 spriteBatch.end();  
  
 this.draw();  
  
 if(debug)  
 drawDebug();  
}

Pemanggilan this.draw pada coding di atas berfungsi untuk me-render actor yang dipanggil pada fungsi buildStage() *(akan dijelaskan pada bagian fungsi buildStage())*.

1. Resize(int w, int h)

Merupakan fungsi turunan dari class Stage. Dalam fungsi ini, dilakukan resize untuk resolusi layar game dan pengambilan titik ukur.

@Override  
public void resize(int w, int h) {  
 this.width = w;  
 this.height = h;  
 ppuX = (float)width / *CAMERA\_WIDTH*;  
 ppuY = (float)height / *CAMERA\_HEIGHT*;  
 Log.*d*("HEIGHT", Float.*toString*(ppuX));  
 Log.*d*("WIDTH", Float.*toString*(ppuY));  
}

ppuX(pixels per unit X) dan ppuY(pixels per unit Y) merupakan titik ukur yang digunakan untuk mempermudah penempatan posisi objek pada layar.

1. loadTextures()

Sesuai namanya, loadTextures digunakan untuk menginisialisasi gambar dan sprite yang ada pada GameScreen. Texture yang di load diantaranya texture untuk state cowboy(IDLE, SHOOT, DIE, WIN) dan background. Texture untuk cowboy dibagi menjadi dua yakni texture untuk cowboy di posisi kiri dan di posisi kanan.

private void loadTextures(){  
 *backgroundTexture* = new Texture(Gdx.*files*.internal("img/game-bg.jpg"));  
 *backgroundSprite* = new Sprite(*backgroundTexture*);  
  
 TextureAtlas atlas = new TextureAtlas(Gdx.*files*.internal("cowboy.pack"));  
 cowboyIdleLeft = atlas.findRegion("standing-sprite");  
 cowboyIdleRight = new TextureRegion(cowboyIdleLeft);  
 cowboyIdleRight.flip(true, false);  
 cowboyHideLeft = atlas.findRegion("hide-sprite");  
 cowboyHideRight = new TextureRegion(cowboyHideLeft);  
 cowboyHideRight.flip(true, false);  
  
 TextureRegion[] shootingRightFrames = new TextureRegion[4];  
 TextureRegion[] winRightFrames = new TextureRegion[10];  
 TextureRegion[] shootingLeftFrames = new TextureRegion[4];  
 TextureRegion[] winLeftFrames = new TextureRegion[10];  
 TextureRegion[] dieRightFrames = new TextureRegion[9];  
 TextureRegion[] dieLeftFrames = new TextureRegion[9];  
  
 for(int i = 0; i < 4; i++){  
 shootingRightFrames[i] = atlas.findRegion("shooting-sprite", (i+1));  
 }  
 shotRightAnimation = new Animation(0.1f, shootingRightFrames);  
  
 for(int i = 0; i < 4; i++){  
 shootingLeftFrames[i] = new TextureRegion(shootingRightFrames[i]);  
 shootingLeftFrames[i].flip(true, false);  
 }  
 shotLeftAnimation = new Animation(0.1f, shootingLeftFrames);  
  
  
 for(int i = 0; i < 10; i++){  
 winRightFrames[i] = atlas.findRegion("win-sprite", (i+1));  
 }  
 winRightAnimation = new Animation(0.2f, winRightFrames);  
 for(int i = 0; i < 10; i++){  
 winLeftFrames[i] = new TextureRegion(winRightFrames[i]);  
 winLeftFrames[i].flip(true,false);  
 }  
 winLeftAnimation = new Animation(0.2f, winLeftFrames);  
  
 for(int i = 0; i < 9; i++){  
 dieRightFrames[i] = atlas.findRegion("dead-sprite", (i+1));  
 }  
 dieRightAnimation = new Animation(0.1f, dieRightFrames);  
 for(int i = 0; i < 9; i++){  
 dieLeftFrames[i] = new TextureRegion(dieRightFrames[i]);  
 dieLeftFrames[i].flip(true,false);  
 }  
 dieLeftAnimation = new Animation(0.1f, dieLeftFrames);  
}

1. loadSound()

Sesuai namanya, fungsi loadSound digunakan untuk menginisialisasi sound untuk GameScreen. Sound yang digunakan diantaranya hitSound(ketika cowboy mengenai sasaran), missSound(ketika cowboy tidak kena sasaran), reloadSound(suara reload), winSound(ketika cowboy menang).

private void loadSound(){  
 hitSound = Gdx.*audio*.newSound(Gdx.*files*.internal("hit.mp3"));  
 missSound = Gdx.*audio*.newSound(Gdx.*files*.internal("miss.mp3"));  
 reloadSound = Gdx.*audio*.newSound(Gdx.*files*.internal("reload.mp3"));  
 winSound = Gdx.*audio*.newSound(Gdx.*files*.internal("win.mp3"));  
}

1. loadGameMode()

Fungsi loadGameMode digunakan untuk refresh mode game dan menginisialisasi settingan – settingan yang diberikan pada cowboy dan inisialisasi flag sesuai dengan tipe game.

private void loadGameMode(){  
 randomBang = rn.nextInt(3) + 1;  
 winId = 0;  
 for(Cowboy cowboy : world.getCowboys()) {  
 cowboy.setState(Cowboy.State.*IDLE*);  
 cowboy.setFlagShot(false);  
 cowboy.setFlagHit(false);  
 cowboy.setFlagReady(false);  
 }  
  
 if(gameMode.getGameType().equals(GameMode.GameType.*CLASSIC*)) {  
 if (gameMode.getRound() == 1 && !flagFinishShot && flagStartGame) {  
 roundFlag = 1;  
 gameMode.setRound(roundFlag);  
 for (Cowboy cowboy : world.getCowboys()) {  
 cowboy.setScore(0);  
 }  
 } else {  
 if (flagStartGame && flagFinishShot) {  
 if (gameMode.nextRound()) {  
 roundFlag = gameMode.getRound();  
 counterStateTime = 0;  
 winStateTime = 0;  
 stateTime = 0;  
 shootSpeedTime = 0;  
 flagStartGame = false;  
 flagCounterStart = false;  
 flagFinishShot = false;  
 flagStartShot = false;  
 flagDrawShot = false;  
 winId = 0;  
 }else{  
 flagEndGame = true;  
 }  
 }  
 }  
 }else{  
 if (score\_limit == 0 && !flagFinishShot && flagStartGame){  
 score\_limit = gameMode.getScore\_limit();  
 for (Cowboy cowboy : world.getCowboys()) {  
 cowboy.setScore(0);  
 }  
 } else {  
 if(flagStartGame && flagFinishShot){  
 for (Cowboy cowboy : world.getCowboys()) {  
 if(cowboy.getScore() > highest\_score)  
 this.highest\_score = cowboy.getScore();  
 }  
  
 if(this.highest\_score < gameMode.getScore\_limit()) {  
 winStateTime = 0;  
 stateTime = 0;  
 counterStateTime = 0;  
 shootSpeedTime = 0;  
 flagStartGame = false;  
 flagCounterStart = false;  
 flagFinishShot = false;  
 flagStartShot = false;  
 flagDrawShot = false;  
 }else{  
 flagEndGame = true;  
 }  
 }  
 }  
 }  
}

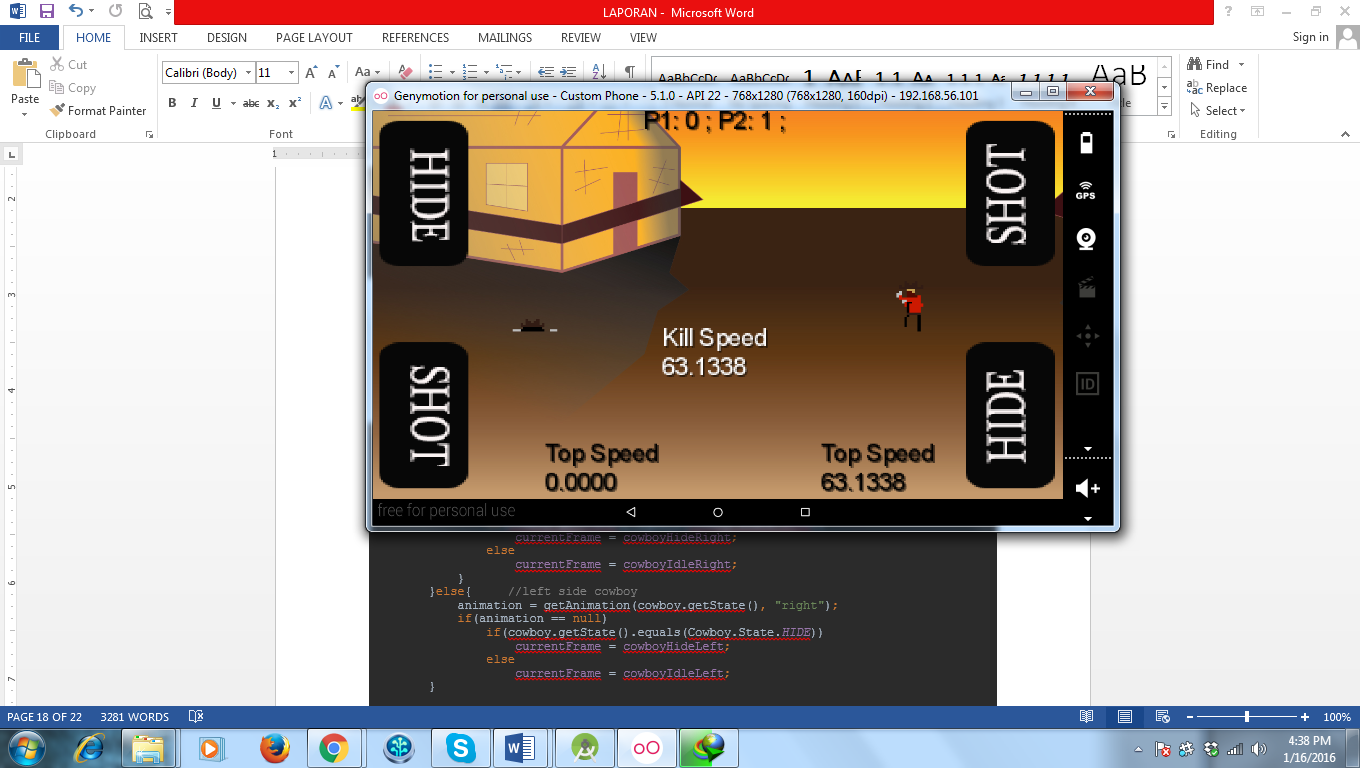
1. getAnimation(CowboyState state, String side)

Fungsi ini digunakan untuk mengembalikan animasi cowboy yang sesuai dengan state dan posisi dari si cowboy(kiri atau kanan). Animasi yang dikembalikan kemudian dapat di jalankan pada fungsi drawCowboy().

private Animation getAnimation(Cowboy.State state, String side){  
 if(state.equals(Cowboy.State.*SHOOT*)){  
 if(side.equals("left"))  
 return shotLeftAnimation;  
 else  
 return shotRightAnimation;  
 }else if(state.equals(Cowboy.State.*LOSE*)){  
 if(side.equals("left"))  
 return dieLeftAnimation;  
 else  
 return dieRightAnimation;  
 }else if(state.equals(Cowboy.State.*WIN*)){  
 if(side.equals("left"))  
 return winLeftAnimation;  
 else  
 return winRightAnimation;  
 }else{  
 return null;  
 }  
}

1. drawCowboy()

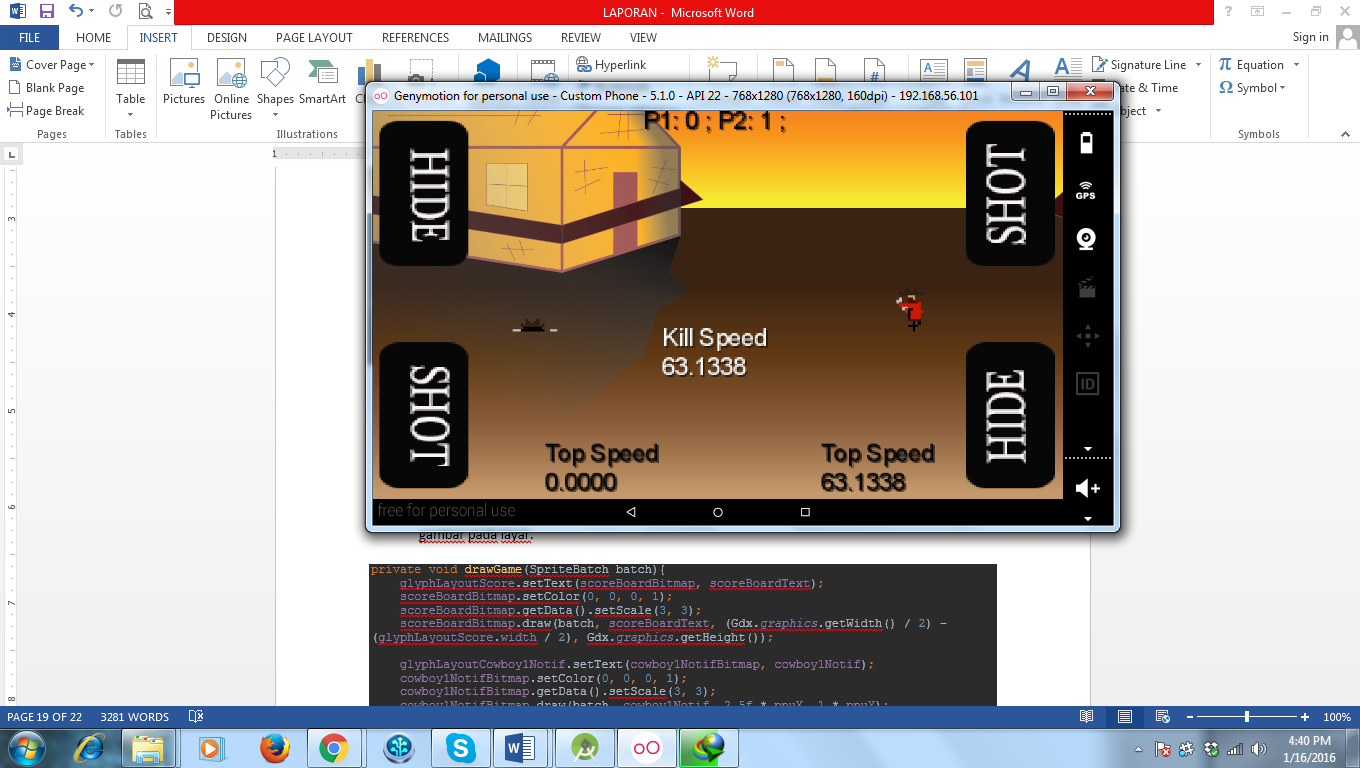
Fungsi ini digunakan untuk menjalankan animasi cowboy. Di dalam fungsi ini diambil timer yang disediakan oleh libGDX untuk mengambil frame animasi yang sesuai. Fungsi ini juga digunakan untuk mengatur pengambilan animasi yang sesuai dengan state dari masing – masing cowboy.



private void drawCowboy(){  
 int i=0;  
 for(Cowboy cowboy : world.getCowboys()){  
 Animation animation;  
 if(i==1) { //right side cowboy  
 animation = getAnimation(cowboy.getState(), "left");  
 if(animation == null) {  
 if(cowboy.getState().equals(Cowboy.State.*HIDE*))  
 currentFrame = cowboyHideRight;  
 else  
 currentFrame = cowboyIdleRight;  
 }  
 }else{ //left side cowboy  
 animation = getAnimation(cowboy.getState(), "right");  
 if(animation == null)  
 if(cowboy.getState().equals(Cowboy.State.*HIDE*))  
 currentFrame = cowboyHideLeft;  
 else  
 currentFrame = cowboyIdleLeft;  
 }  
  
 if (animation != null) {  
 if (flagFinishShot && cowboy.getState().equals(Cowboy.State.*WIN*) && cowboy.getCowboyId() == winId) {  
 winStateTime += Gdx.*graphics*.getDeltaTime();  
 currentFrame = animation.getKeyFrame(winStateTime, true);  
 } else {  
 if (animation.isAnimationFinished(stateTime)) {  
 if (cowboy.isFlagHit() && cowboy.getCowboyId() == winId) {  
 cowboy.setState(Cowboy.State.*WIN*);  
 flagFinishShot = true;  
 stateTime = 0;  
 }else {  
 currentFrame = animation.getKeyFrame(stateTime);  
 }  
 } else {  
 stateTime += Gdx.*graphics*.getDeltaTime();  
 currentFrame = animation.getKeyFrame(stateTime, true);  
 }  
 }  
 }  
  
 spriteBatch.draw(currentFrame, cowboy.getPosition().x \* ppuX, cowboy.getPosition().y \* ppuY,  
 cowboy.getSize() \* ppuX, cowboy.getSize() \* ppuY);  
 i++;  
 }  
}

1. drawGame()

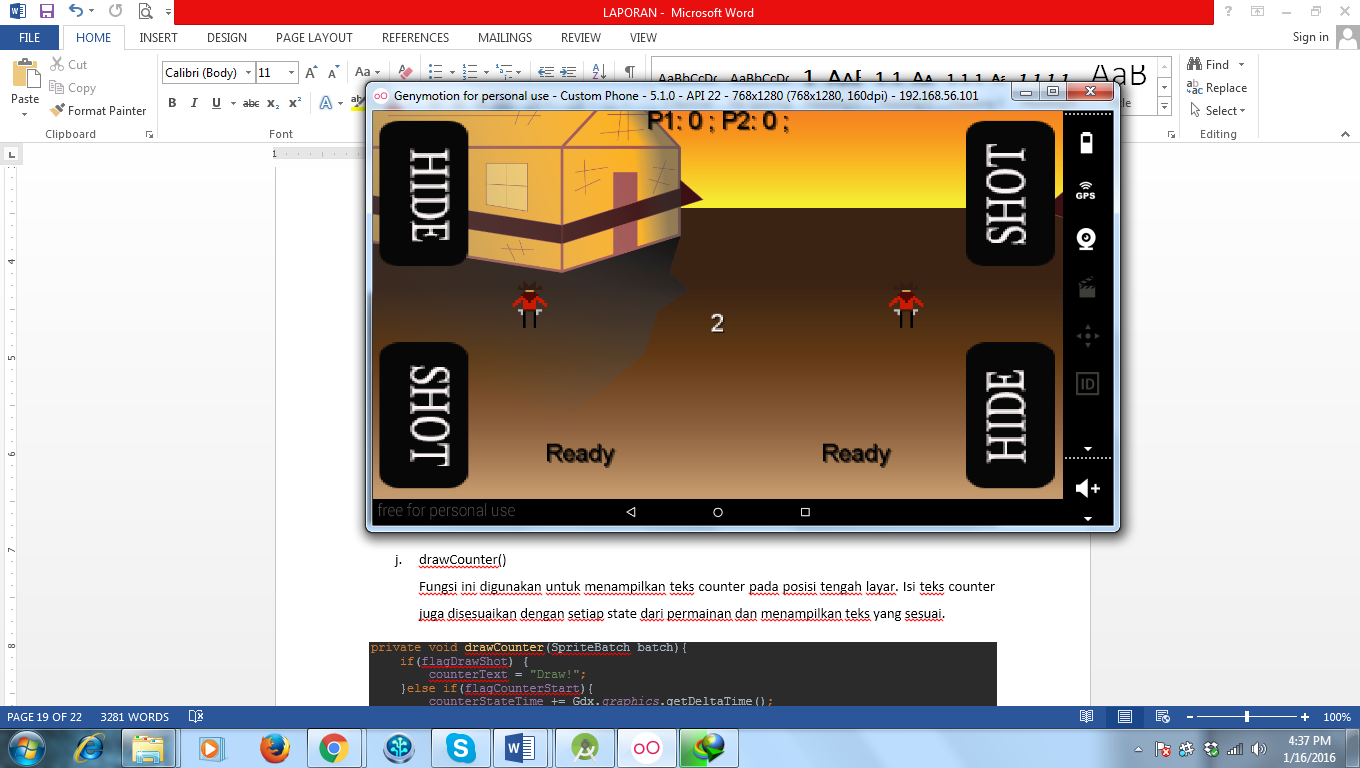
Fungsi ini digunakan untuk menampilkan teks yang berisi informasi game seperti score, ronde, dan kecepatan cowboy. Fungsi ini diletakkan di dalam render sehingga dapat di-refresh terus menerus dan menampilkan teks yang sesuai dengan tipe game. Fungsi ini menerima SpriteBatch yang dimana SpriteBatch merupakan objek yang digunakan untuk meletakkan teksture atau gambar pada layar.



private void drawGame(SpriteBatch batch){  
 glyphLayoutScore.setText(scoreBoardBitmap, scoreBoardText);  
 scoreBoardBitmap.setColor(0, 0, 0, 1);  
 scoreBoardBitmap.getData().setScale(3, 3);  
 scoreBoardBitmap.draw(batch, scoreBoardText, (Gdx.*graphics*.getWidth() / 2) - (glyphLayoutScore.width / 2), Gdx.*graphics*.getHeight());  
  
 glyphLayoutCowboy1Notif.setText(cowboy1NotifBitmap, cowboy1Notif);  
 cowboy1NotifBitmap.setColor(0, 0, 0, 1);  
 cowboy1NotifBitmap.getData().setScale(3, 3);  
 cowboy1NotifBitmap.draw(batch, cowboy1Notif, 2.5f \* ppuX, 1 \* ppuY);  
  
 glyphLayoutCowboy2Notif.setText(cowboy1NotifBitmap, cowboy2Notif);  
 cowboy2NotifBitmap.setColor(0, 0, 0, 1);  
 cowboy2NotifBitmap.getData().setScale(3, 3);  
 cowboy2NotifBitmap.draw(batch, cowboy2Notif, 6.5f \* ppuX, 1 \* ppuY);  
}

1. drawCounter()

Fungsi ini digunakan untuk menampilkan teks counter pada posisi tengah layar. Isi teks counter juga disesuaikan dengan setiap state dari permainan dan menampilkan teks yang sesuai.



private void drawCounter(SpriteBatch batch){  
 if(flagDrawShot) {  
 counterText = "Draw!";  
 }else if(flagCounterStart){  
 counterStateTime += Gdx.*graphics*.getDeltaTime();  
 if(Math.*ceil*(counterStateTime) <= 3) {  
 counterText = Integer.*toString*((int) (Math.*ceil*(counterStateTime)));  
 }else if(Math.*ceil*(counterStateTime) == (randomBang + 3)) {  
 counterText = "DOR !";  
 flagCounterStart = false;  
 flagStartShot = true;  
 counterStateTime = 0;  
 cowboy1Notif = "";  
 cowboy2Notif = "";  
 }else {  
 counterText = "";  
 }  
 }  
  
 if(!flagStartGame){  
 counterText = "Press Shot Button to Start";  
 }else if(flagFinishShot){  
 counterText = "Kill Speed \n";  
 for(Cowboy cowboy : world.getCowboys()) {  
 DecimalFormat df = new DecimalFormat("0.0000");  
 if(cowboy.getCowboyId() == winId){  
 counterText+= df.format(cowboy.getSpeed());  
 }  
  
 if (cowboy.getCowboyId() == 1)  
 cowboy1Notif = "Top Speed \n"+df.format(cowboy.getTopSpeed());  
 else  
 cowboy2Notif = "Top Speed \n"+df.format(cowboy.getTopSpeed());  
 }  
 }  
  
 if(flagStartShot)  
 shootSpeedTime+= Gdx.*graphics*.getDeltaTime();  
  
 glyphLayoutScore.setText(counterBitmap, counterText);  
 counterBitmap.setColor(1, 1, 1, 1);  
 counterBitmap.getData().setScale(3, 3);  
 counterBitmap.draw(batch, counterText, (Gdx.*graphics*.getWidth() / 2) - (glyphLayoutScore.width / 2),  
 (Gdx.*graphics*.getHeight() / 2) - (glyphLayoutScore.height / 2));  
}

1. drawBackground()

Sesuai namanya, fungsi ini hanya digunakan untuk menampilkan background. Fungsi ini juga diletakkan di bagian render

private void drawBackground(SpriteBatch batch){  
 *backgroundSprite*.draw(batch);  
}

1. drawDebug()

Fungsi ini digunakan untuk menampilkan garis debug pada tiap objek. Digunakan hanya untuk tujuan debugging pada saat pertama kali mencoba membuat game dan mengetahui posisi – posisi cowboy yang sesuai.

private void drawDebug(){  
 //render Cowboy  
 debugRenderer.setProjectionMatrix(cam.combined);  
 debugRenderer.begin(ShapeRenderer.ShapeType.*Filled*);  
 for(Cowboy cowboy : world.getCowboys()){  
 Rectangle rect = cowboy.getBounds();  
 float x1 = cowboy.getPosition().x + rect.x;  
 float y1 = cowboy.getPosition().y + rect.y;  
 debugRenderer.setColor(new Color(1, 0, 0, 1));  
 debugRenderer.rect(x1, y1, rect.width, rect.height);  
 }  
 debugRenderer.end();  
}

1. getTouchedCowboy()

Fungsi ini menerima satu parameter yakni touchId.TouchId ini hanyalah sebuah angka yang di set pada masing – masing cowboy untuk mempermudah identifikasi. Fungsi ini digunakan untuk melooping array cowboy dan mengembalikan cowboy yang sesuai dengan touchId. Fungsi ini digunakan pada fungsi buttonTouch().

private Cowboy getTouchedCowboy(int touchId){  
 for (Cowboy cowboy : world.getCowboys()) {  
 if(cowboy.getCowboyId() == touchId){  
 return cowboy;  
 }  
 }  
 return null;  
}

1. getEnemyCowboy()

Mirip dengan fungsi getTouchedCowboy, hanya saja fungsi ini mengembalikan cowboy yang tidak sesuai dengan touchId dan diasumsikan bahwa cowboy tersebut adalah cowboy milik lawan.

private Cowboy getEnemyCowboy(int touchId){  
 for (Cowboy cowboy : world.getCowboys()) {  
 if(cowboy.getCowboyId() != touchId){  
 return cowboy;  
 }  
 }  
 return null;  
}

1. buttonTouch()

Fungsi ini merupakan salah satu fungsi yang cukup penting. Fungsi ini akan mem-filter semua kondisi flag dan menghasilkan parameter – parameter yang sesuai dengan kondisi game. Setiap kondisi didasari dengan flag – flag yang berbeda dan memastikan tombol yang disentuh pada kondisi tertentu merupakan tombol yang sesuai. Fungsi ini juga mengembalikan flag yang akan menghasilkan teks yang berbeda untuk dirender pada layar game.

private void buttonTouch(int touchId, String buttonType){  
 Cowboy touchedCowboy = getTouchedCowboy(touchId);  
 Cowboy enemyCowboy = getEnemyCowboy(touchId);  
 if(flagEndGame){  
 ScreenManager.*getInstance*().showScreen(ScreenEnum.*LEVEL\_SELECT*, null);  
 }else if(flagDrawShot){  
 loadGameMode();  
 reloadSound.play();  
 flagDrawShot = false;  
 flagFinishShot = true;  
 }else if(!flagStartGame) {  
 if(buttonType.equals("shoot")){  
 if(!touchedCowboy.isFlagReady()){  
 reloadSound.play();  
 touchedCowboy.setFlagReady(true);  
 }  
  
 if(touchedCowboy.isFlagReady() && enemyCowboy.isFlagReady()){  
 bothCowboyReady = true;  
 flagStartGame = true;  
 flagCounterStart = true;  
  
 if(touchedCowboy.isFlagReady()) {  
 if (touchedCowboy.getCowboyId() == 1)  
 cowboy1Notif = "Ready";  
 else  
 cowboy2Notif = "Ready";  
 }  
  
 loadGameMode();  
 }  
  
 if(touchedCowboy.isFlagReady()) {  
 if (touchedCowboy.getCowboyId() == 1)  
 cowboy1Notif = "Ready";  
 else  
 cowboy2Notif = "Ready";  
 }  
 }  
 }else if(flagFinishShot){  
 loadGameMode();  
 scoreBoardText = "";  
 for (Cowboy cowboy : world.getCowboys()) {  
 scoreBoardText += "P" + cowboy.getCowboyId() + ": " + cowboy.getScore() + " ; ";  
 }  
 if (gameMode.getGameType().equals(GameMode.GameType.*CLASSIC*))  
 scoreBoardText += "Round: " + gameMode.getRound();  
 }else if(flagStartShot){  
 scoreBoardText="";  
 if(winId == 0){  
 if(buttonType.equals("hide")){  
 touchedCowboy.setState(Cowboy.State.*HIDE*);  
 touchedCowboy.setSpeed(shootSpeedTime);  
 }else if(buttonType.equals("idle")) {  
 touchedCowboy.setState(Cowboy.State.*IDLE*);  
 touchedCowboy.setSpeed(0);  
 }else if(buttonType.equals("shoot")){  
 if (!touchedCowboy.isFlagShot() && touchedCowboy.getState().equals(Cowboy.State.*IDLE*)) { //Cowboy who follow rule  
 touchedCowboy.setState(Cowboy.State.*SHOOT*);  
 touchedCowboy.setFlagShot(true);  
 touchedCowboy.setSpeed(shootSpeedTime);  
  
 if(enemyCowboy.getState().equals(Cowboy.State.*IDLE*)){  
 touchedCowboy.addScore();  
 winId = touchId;  
 hitSound.play();  
 winSound.play();  
 touchedCowboy.setFlagHit(true);  
 enemyCowboy.setState(Cowboy.State.*LOSE*);  
  
 Log.*d*("touched speed", Float.*toString*(touchedCowboy.getSpeed()));  
 if(touchedCowboy.getTopSpeed() == 0)  
 touchedCowboy.setTopSpeed(touchedCowboy.getSpeed());  
 else if(touchedCowboy.getSpeed() < touchedCowboy.getTopSpeed())  
 touchedCowboy.setTopSpeed(touchedCowboy.getSpeed());  
  
 }else if(enemyCowboy.getState().equals(Cowboy.State.*SHOOT*)){  
 if(touchedCowboy.getSpeed() < enemyCowboy.getSpeed() && enemyCowboy.getSpeed()!= 0) {  
 touchedCowboy.addScore();  
 winId = touchId;  
 hitSound.play();  
 winSound.play();  
 touchedCowboy.setFlagHit(true);  
 enemyCowboy.setState(Cowboy.State.*LOSE*);  
 if (touchedCowboy.getTopSpeed() == 0)  
 touchedCowboy.setTopSpeed(touchedCowboy.getSpeed());  
 else if (touchedCowboy.getSpeed() < touchedCowboy.getTopSpeed())  
 touchedCowboy.setTopSpeed(touchedCowboy.getSpeed());  
 }else if(touchedCowboy.getSpeed() > enemyCowboy.getSpeed() && !enemyCowboy.isFlagHit() && enemyCowboy.isFlagShot()){  
 touchedCowboy.addScore();  
 winId = touchId;  
 hitSound.play();  
 winSound.play();  
 touchedCowboy.setFlagHit(true);  
 enemyCowboy.setState(Cowboy.State.*LOSE*);  
 if (touchedCowboy.getTopSpeed() == 0)  
 touchedCowboy.setTopSpeed(touchedCowboy.getSpeed());  
 else if (touchedCowboy.getSpeed() < touchedCowboy.getTopSpeed())  
 touchedCowboy.setTopSpeed(touchedCowboy.getSpeed());  
 }  
 }else if(enemyCowboy.getState().equals((Cowboy.State.*HIDE*))){  
 missSound.play();  
 if(touchedCowboy.isFlagShot() && enemyCowboy.isFlagShot() && winId == 0){  
 flagDrawShot = true;  
 }  
 }  
 }  
 }  
 }  
  
 for (Cowboy cowboy : world.getCowboys()) {  
 scoreBoardText += "P" + cowboy.getCowboyId() + ": " + cowboy.getScore() + " ; ";  
 }  
  
 if(gameMode.getGameType().equals(GameMode.GameType.*CLASSIC*))  
 scoreBoardText += "Round: " + gameMode.getRound();  
 }  
}

1. buildStage()

Fungsi buildStage pada class GameScreen hanya digunakan untuk menambahkan button untuk pemain 1 dan 2. Masing – masing pemain mendapatkan dua buah tombol yakni Shot dan Hide. Tombol – tombol ini masing – masing juga memiliki listener yang dimana setiap aksinya akan dikirimkan ke fungsi buttonTouch untuk kemudian diberikan respon yang sesuai dengan tombol yang disentuh. Dikarenakan kami telah meng-override fungsi render, tombol – tombol tersebut tidak akan bisa muncul pada layar game dikarenakan objek yang diletakkan pada fungsi buildStage ini akan langsung digantikan oleh objek baru yang di render. Oleh karena itu, pada fungsi render() kami meletakkan pemanggilan draw.

this.draw();

Pemanggilan draw ini akan memastikan objek – objek yang diletakkan pada fungsi buildStage juga di render kembali pada fungsi render(). Pemanggilan this.draw ini diletakkan setelah pemanggilan fungsi lainnya dengan tujuan tombol yang dihasilkan dapat diklik dan tidak tertimpa oleh gambar – gambar lain.

@Override  
public void buildStage() {  
 float ppux = 320.0f / cam.viewportWidth;  
 float ppuy = 240.0f / cam.viewportHeight;  
 ImageButton btnShotLeft = UIFactory.*createButton*(new Texture(Gdx.*files*.internal("img/shot-left.png")));  
 btnShotLeft.setPosition(0, 0 \* ppuy);  
 btnShotLeft.setSize(1.5f\*ppux, 3\*ppuy);  
 addActor(btnShotLeft);  
  
 ImageButton btnHideLeft = UIFactory.*createButton*(new Texture(Gdx.*files*.internal("img/hide-left.png")));  
 btnHideLeft.setPosition(0, 4 \* ppuy);  
 btnHideLeft.setSize(1.5f \* ppux, 3 \* ppuy);  
 addActor(btnHideLeft);  
  
 ImageButton btnShotRight = UIFactory.*createButton*(new Texture(Gdx.*files*.internal("img/shot-right.png")));  
 btnShotRight.setPosition((cam.viewportWidth - 1.5f) \* ppux, 4 \* ppuy);  
 btnShotRight.setSize(1.5f \* ppux, 3 \* ppuy);  
 addActor(btnShotRight);  
  
 ImageButton btnHideRight = UIFactory.*createButton*(new Texture(Gdx.*files*.internal("img/hide-right.png")));  
 btnHideRight.setPosition((cam.viewportWidth - 1.5f) \* ppux, 0 \* ppuy);  
 btnHideRight.setSize(1.5f \* ppux, 3 \* ppuy);  
 addActor(btnHideRight);  
  
 btnShotLeft.addListener(new InputListener() {  
 @Override  
 public boolean touchDown (InputEvent event, float x, float y, int pointer, int button) {  
 buttonTouch(1, "shoot");  
 return true;  
 }  
 });  
 btnShotRight.addListener(new InputListener() {  
 @Override  
 public boolean touchDown (InputEvent event, float x, float y, int pointer, int button) {  
 buttonTouch(2, "shoot");  
 return true;  
 }  
 });  
  
 btnHideLeft.addListener(new InputListener() {  
 @Override  
 public boolean touchDown (InputEvent event, float x, float y, int pointer, int button) {  
 buttonTouch(1, "hide");  
 return true;  
 }  
  
 @Override  
 public void touchUp (InputEvent event, float x, float y, int pointer, int button) {  
 buttonTouch(1, "idle");  
 }  
 });  
 btnHideRight.addListener(new InputListener() {  
 @Override  
 public boolean touchDown (InputEvent event, float x, float y, int pointer, int button) {  
 buttonTouch(2, "hide");  
 return true;  
 }  
  
 @Override  
 public void touchUp (InputEvent event, float x, float y, int pointer, int button) {  
 buttonTouch(2, "idle");  
 }  
 });  
}

# References & Credits

Angel, 2014. *How to manage screens in LibGDX.* [Online]   
Available at: http://www.pixnbgames.com/blog/libgdx/how-to-manage-screens-in-libgdx/  
[Accessed 31 12 2015].

bss9, 2014. *Extending the simple game.* [Online]   
Available at: https://github.com/libgdx/libgdx/wiki/Extending-the-simple-game  
[Accessed 30 12 2015].

gameFromScratch.com, 2013. *LIBGDX TUTORIAL 3B: SIMPLE ANIMATION.* [Online]   
Available at: http://www.gamefromscratch.com/post/2013/12/09/LibGDX-Tutorial-3B-Simple-Animation.aspx  
[Accessed 31 12 2015].