**Shoot the Goomba’s – Design Document**

*List of database tables and fields*

High Score table in SQLlite with two fields:

* name: TEXT contains the name of the user
* points: INT contains the number of points the user has scored

SharedPreferences in a XML file “*goombasPrefs*”

* name: String. Contains the name of the user
* music: boolean. True if the user has chosen the music to be on.

Configuration file in config.properties, the defined values are:

* numberOfBullets - number of bullets on reload
* gameTime – time the user has to play in milliseconds
* values of the Goomba’s
* flyDurations of the Goomba’s
* size of the Goomba’s

*List of classes and methods*

MainActivity: Controls the home page.

* public void startGame(View view): Called when the Start button in the view is clicked. Starts an intent to the gameActivity class with the extra value it’s a new game.
* public void highScores(View view): Called when the high Scores button in the view is clicked. Starts an intent to the scoreActivity class.

gameActivity: Controls the page where the game is played

* protected void onCreate(Bundle savedInstanceState):
  + Hides the status and action bar.
  + Gets the Shared Preferences Object.
  + Sets the countdown timer,
  + Start a new instance of the class GamePlay,
  + Loads the music button,
  + Sets the target, make sure it starts outside the view
  + Loads the bullets
  + Set number of points in the top right of the screen
* Private void setTimer() : Sets a countdown timer of 60 seconds and calls the addGoomba function every second, calls the addClock function every second with a probability of 1/25. Finishes the game when the time is up.
* Private void addGoomba(); Adds a new Goomba to the view and sets the target to the coordinates of the goomba when it is shot.
* Private void addClock(); Adds a new extraTimeClock to the view, when the user clicks it, the user gets 5 extra seconds play time.
* Private void insertName(); Asks user for name input to add to the High Score table.
* Public Boolean onTouchEvent(MotionEvent event); Handles touches on the screen.
* Public void loadBullets(View view); Called when the user clicks the “Reload” button.
* Private void addReloadButton(); Add ReloadButton to the view.
* Protected void onStop(); Finish activity when game is sent to the background.

scoreActivity: Controls the page where the high scores are shown.

* Protected void onCreate(Bundle savedInstanceState);
  + Hides the status and action bar
  + Shows the text “High Scores” if the user is directed from the home page and the text “You got .. points” is directed from the game page.
  + Shows a table with 5 high scores
* Public void showData(); show the high scores in a table to the user.
* Protected void onStop(); Finish the activity when the game is send to the background.

gamePlay: Controls the view for the Goomba game.

* field variables
  + private final Context myContext;
  + private SharedPreferences prefs
  + private int points : number of points scored by in the game
  + private int gameTime: time the user has on start to play the game
  + private int numberOfBullets : number of bullets left in the game
  + private boolean music : true if music is on
  + private MediaPlayer mpReload : sound for reloading bullets
  + private MediaPlayer mpGoomba : sound for shooting a goomba
  + private MediaPlayer mpEmpty : sound for shooting without bullets
  + private MediaPlayer mpBackground : background music
* constructor
  + this.myContext = context;
  + this.prefs = preferences;
  + this.points = 0 : set scored points to zero
  + this.numberOfBullets =: set number of bullets as defined in the config file
  + this.gameTime = : set time in milliseconds as defined in the config file
  + this.music = prefs.getBoolean("music", true) : get value of music from SharedPreferences
  + mpReload = MediaPlayer.create(myContext, R.raw.gun\_cocking\_01);
  + mpGoomba = MediaPlayer.create(myContext, R.raw.smw\_stomp);
  + mpEmpty = MediaPlayer.create(myContext, R.raw.gun\_empty);
  + mpBackground = MediaPlayer.create(myContext, R.raw.super\_mario\_bros);
  + mpBackground.setVolume(0.0f, 0.8f);
  + mpBackground.start();
* Methods
  + switchMusic : change the value of the music from on/off
  + playEmpty: play the sound of an empty bullet
  + playShotGoomba: play the sound of a shot goomba
  + playReload: play the sound of reloading a bullet
  + muteBackground: set the volume of the background music to zero
  + stopMusic: stop the background music
  + updatePoints: update the number of scored points
  + updateBullets: Update the number of used bullets
  + getNumberOfBullets :
  + getMusicValue
  + getPoints
  + getGameTime

Goomba: class for a goomba, extends the button class

* Field Variables
  + Int screenY: height of the screen
  + Int screenX: length of the screen
  + Int value: value of the goomba
  + Int flyDuration: milliseconds it takes the goomba to fly from one side to the other
  + Int size: size of the Goomba
  + Int y : Starting height of the goomba
  + Boolean fromLeft : true if the goomba starts from left
  + AnimatorSet animSet: Animator Set for the movement of the Goomba
  + Context myContext;
* Constructor
  + Calculate the screenY and screenX
  + Gets the value, fly duration and size from the config file
  + Picks a random starting height
  + Gets a random Boolean to determine if the Goomba starts from the left or right side.
  + Enable the default sound of the Goomba
* Methods
  + Private void SetY(View goomba); Sets the starting height of the Goomba.
  + Public void startAnimation(); Starts the movement of the Goomba.
  + Public void shot(); Called when the Goomba is clicked, changes the picture of the Goomba and enables the user to click the Goomba again.
  + Public void getValue() ; returns the value of the Goomba

DBAdapter: Class for the Database adapter to handle actions with the high score database.

* Field Variables
  + String KEY\_; Columns of the High Score table: id, name, points
  + String / int DB\_; Fields for the Database: name of the database file, name of the table and version number
  + String DATABASE\_CREATE; SQL statement for creating a database
  + DatabaseHelper dbHelper; helper for the database to handle the creation and update of a table
  + SQLiteDatabase db; the SQLite database
  + Context myContext
* Constructor
* Methods
  + public boolean checkHighScore( int points); Return true if last played game is a high score
  + public Cursor findScores(); find the current list of high scores
  + public SQLiteDatabase insertScore(String name, int points); Insert a new high score in the database
  + public String getName(Cursor cursor); Returns the name of the row selected by the cursor
  + public String getPoints (Cursor cursor); Returns the points of the row selected by the cursor

Bullet: Class for the image of a single bullet, extends the Image View class

* Constructor
* Methods
  + setBulletParameters(int i); Sets the parameters for the location of the bullet in the screen and an id and image to it.

ReloadButton: Class for the Reload Button, extends the Button class

* Field Variables
  + Context myContext
* Constructor
  + Defines the context
* Methods
  + public void onAppearance(GamePlay game); Play a sound and vibrate when the Reload Button appears.
  + Public void setReloadParams();Sets the parameters for the location of the button, add text and enable the default sound.

ShowValue: Class for values shown when a goomba is shot

* Field Variables
  + Protected int x; x-coordinate of the show value
  + Protected int y; y-coordinate of the show value
* Constructor
  + Sets the x and y coordinates of the show value
* Methods
  + public void setValueParams(int size, String value); Sets the parameters for the location of the Show Value and sets its text to the value.
  + Public void valueAnimation(); Adds an animation to the show value.

Target: Class for the target, extends the ImageView.

* Field Variables
  + Protected int size: size of the target
  + Protected RelativeLayout.LayoutParams targetParams; parameters for the target
* Constructor
  + Defines the size of the target and set parameters for the size.
* Methods
  + Public void setTargetParams(int x, int y); Sets parameters for the location of the target to x and y.
  + Public int getSize(): returns the size of the target.

ExtraTimeClock

* Field Variables
* Contstructor
* Methods
  + Public void setAnimation(); Set an animation for moving the clock down the screen in 4 seconds.

AssetsPropertyReader: Helper class for reading the config.properties file, Source: http://khurramitdeveloper.blogspot.nl/2013/07/properties-file-in-android.html

* Field Variables
  + private Context context;
  + private Properties properties;
* Constructor
  + Adds a new property object
* Methods
  + public Properties getProperties(String FileName); Returns the properties from the file with the filename from the input.