24005044

Android Game Report – Golden Bird

2nd Major Coursework for Java CS2JA16

Contents

Abstract

This report is documentation for the creation of an android game. Detailed from the design stage to the final implementation stage and every stage in between. Showcasing explores the game at the highest level which the general user would interact with, going further down to the optimisation of code and memory management and how these can be utilised within android to keep things running smoothly and without a crash. Included inside are extension tasks and how these enhance the game to make the user experience better.

Chapter 1: Introduction and Showcase

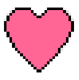
This aim of project was to create and implement a Java based Android game. The game was required to start on a separate screen to that of the gameplay and function with the android library buttons. Display a current score of the player, the scoring mechanics were to be chosen and designed by the developer. Have a player controlled character that could move in at least both the x and y planes, also the player character must have a life or health system allowing the player to take damage and eventually lose the game. Have at least 3 game levels, which can either be hard coded or be randomly generated. Finally, the game must have multiple opponents that the player is working to beat, dodge or compete with.

In the following chapters, there will be discussion of OOP design and why it is important to this project, Android specific memory usage and speed improvements and overall potential improvements to the current game state and design of the game.

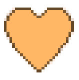
Game Showcase:

Entity Types:

Heart – Plus 1 Life



Heart – GUI life indicator



Coin – Plus 10 points

Speed Boost – Increases speed in current direction

Enemy Destroyer – Destroys some of the current UFO enemies on screen

UFO – Enemy character that falls to the ground whilst bouncing off the walls

Balloon – Immovable object used during random level design

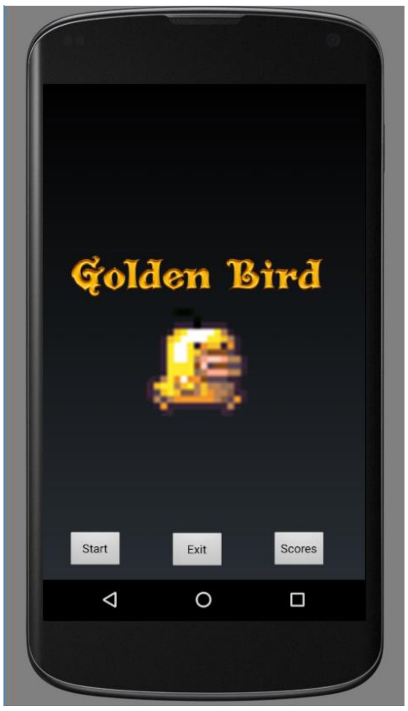
Player Character – Player controllable character

Gameplay:

The aim of the game is to collect the randomly spawning coins for points whilst avoiding the enemies falling from the sky looking to take your lives away. During the level, there is a timer running named secondsEllapsed, this is used to define a multiplier, which when enemies spawn the spawn rates are timed by the multiplier to ensure a gradual difficulty curve the longer the player survives. Whilst playing there are 2 types of power ups; Speed Boost and Enemy Destroyer. The Speed Boost is simply a speed enhancer for the player allowing them to accelerate across the screen towards awaiting coins. The Enemy Destroyer has a chance to eliminate any enemies currently on the screen at which point the player receives 5 points per enemy destroyed by this method. On collision of the player with an enemy the player loses 1 life per collision and receives a 5-point deduction along with a speed reduction of half after the collision. Once the player's life count becomes 3 or lower Hearts will spawn into the game allowing the player to recover 1 life per heart collected, up to a maximum of 5 lives at any time.

Once the player reaches 0 remaining lives the game ends and the resulting score is the uploaded to FireBase as an online high score database as Google Play Services was restricted for this particular project.

Main Menu Screen



Above the main menu screen can be seen, it is simplistic yet effective.

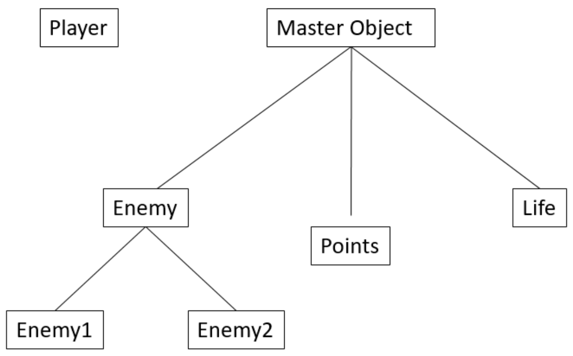
Chapter 2: Object Orientated Design

Introduction

As explained in the introduction the aim of this project was to create a Java based Android game for the android operating system, with a base SDK of 21. During the design phase of the game a few key things had to be taken into consideration: the engagement of the game, complexity and simplicity and how well it will run on the target device.

In terms of requirements specifically needed for android development there were a few that were needed. The code that was written had to be lightweight and fast as a mobile processor isn't as fast or as capable as that of a pc. This was a hindrance as everything needed to be properly optimised to ensure smooth running. Another special feature needed for android development is the ability to utilise XML as the activity views are primarily coded in XML.

Method



Above is my initial design for the project.

The initial design that was created had one master object which would have an x, y, xspeed, yspeed and an image. This aided with the creation due to inheritance being used. This allows for the same code to be used and be reused by multiple children classes.

The initial design was purposely very sparse to allow extra development as and when it was needed. It was decided that this was the best way because as the game was being constantly developed and tested new innovative ideas came around and were implemented. Once implemented not all of the features were kept as it was felt that the game either lacked direction or was becoming too complex.

Discussion

Initially there were only a handful of classes. Master object which would be used to abstract into the child classes Enemy, Points and Life. The Master object would initially be constructed with an x value, y value, x speed and a y speed. At the end of the project more children had been created as development had progressed during the game and more object were needed to make the game enjoyable and playable. With this came the introduction of the static balloon obstacle which has bouncing properties and a death power up that when the player collides with has a chance to destroy some of the enemy objects. Also as a power up a speed power up was added which when a collision is detected the player has a speed multiplier in the current direction of travel.

Conclusions

The most important design change was when the decision was finally made to create the game in the style it was created in. This lent itself very well to OOP. In a similar way to the first piece of Java coursework I setup my classes to inherit from the parent class. The children classes then have child specific behaviour which depends upon the assigned image. The image is what is drawn to screen every tick and denotes the type of object. Within my TheGame class specific collision detection is implemented and run when objects collide. The results can be object deletion or object manipulation.

Chapter 3: Memory Usage and Speed Improvements

Introduction

Android devices are a lot less powerful than that of desktop and laptops. Therefore, during development, the developer must be careful with their solutions and memory consumption. During Android development (version 5.0 Lollipop) triggering the garbage collector could result in delays to the user of 200ms, which if they are surfing the web and scrolling though lists this is a noticeable delay. A way of saving memory is to reuse objects instead of creating new ones and destroying unused ones.

Another way that memory can quickly become full is when loading Bitmaps. If the required bitmap is 400dpx400dp but the size being displayed is only 100dpx100dp then the true smaller size should be kept in memory. It doesn't seem like a lot, however when there are many Bitmaps being held in memory having unneeded pixels can attribute to the slowing of the system.

A more advanced way of improving memory and efficiency is to use LRU Cache. A LRU Cache keeps track of the usage of its members. For example, if there are a number of objects and they are being accessed sequentially, each object will be put in the cache and when the get() function is called the object is moved to the top of the cache. Object that stay at the bottom of the cache for the longest time are removed from cache as they are most likely unneeded and not going to be accessed any time soon.

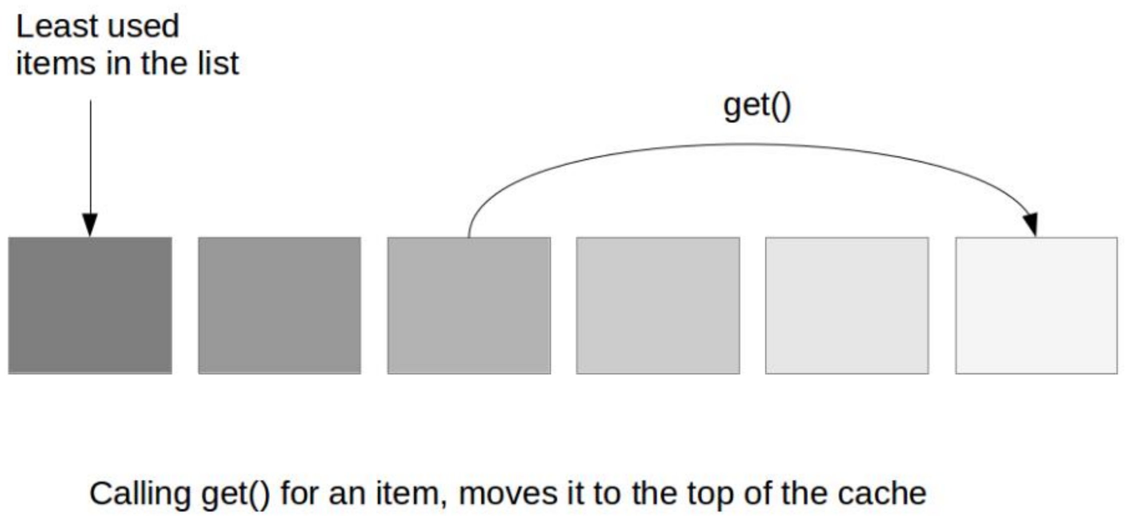


Image showing a visual representation of LRU Cache
(Image from: http://www.vogella.com/tutorials/AndroidApplicationOptimization/article.html)

Method

To improve the main thing that was implemented was that I loaded Bitmaps in at the correct size as opposed to a wasteful size that I would never use. The way I did this was by recording my emulator on my desktop with Open Broadcasting Software (OBS) which has a built in frame per second counter. Since I have used this software before alongside my GPU software to record frames per second I am happy to trust the reading from OBS. I recorded the entire game from start to all lives lost. I then took the average over the play time period with always the number of entities present.

Another smaller yet important improvement that was made during development was using a GameObject instead of using Object.get(i) each time. The main reasons for doing this was that that code looked more asthetic and also that sometimes a null pointer exception would be thrown for a random reason.

Result

Number of entity	Bitmap Size	AVG Frame per second
10	Not Scaled	48
10	Scaled	60
50	Not Scaled	25
50	Scaled	40
100	Not Scaled	20
100	Scaled	30
500	Not Scaled	10
500	Scaled	20
1000	Not Scaled	5
1000	Scaled	15

These results are those of a “stress” test type of test. To lower the frames per second of the game I artificially created the desired amount of entities on start up to allow for the bitmap scaled and non-scaled to be shown. These

Discussion

The improvements that were made a significant difference the larger the number of entity bitmaps being loaded into memory. At 1500 bitmaps between scaled and non-scaled there was a 50% reduction in framerate, this made the game noticeably laggy. Due to this lag the player couldn’t move as intended and died almost instantly. In the non-scaled tests the greater the number of unscaled pictures the greater the decrease of frames, this was also in a logarithmic style which made the decay in frames even more noticeable.

This was great as even though there was still a substantial frame rate loss due to the huge number of bitmaps being loaded the drops in frames wasn’t as bad as non-scaled bitmaps. Testing the number of bitmaps up to 1000 is extreme testing as during a play through of the game there would never be an instance where that number of objects would be spawning at once. A more realistic number to look at would be 50 entities. 25 fps not-scaled and 40 scaled. Just by scaling the images there has been a 60% increase in frame rate.

Conclusions

Practically the improvement that was made contributed to a 60% improvement in frames per second. In a less practical situation (1000 entities spawning) there was a 200% increase in frames per second. This is a huge difference and indicates the importance on not wasting memory on useless bytes of a bitmap or in any part of writing code.

Chapter 4: Improvement or extension

Introduction

In comparison to other android games out currently, the game that has been developed isn't much like any others. In an obscure way, the game is a reminder of 'Break Out', flappy bird and an old-school gold collecting game. The reason I saw this type of game fit to develop is that there aren't many games of this type around. Also I tried to incorporate a simplistic element to the game to ensure game play didn't last for too long per game as mobile gaming is meant to be fast and on the go. By implementing a system that causes spawn rates of enemies to increase based upon the secondsEllapsed variable allows for the game to ramp up in difficulty and potentially surprise the player whilst playing.

Method

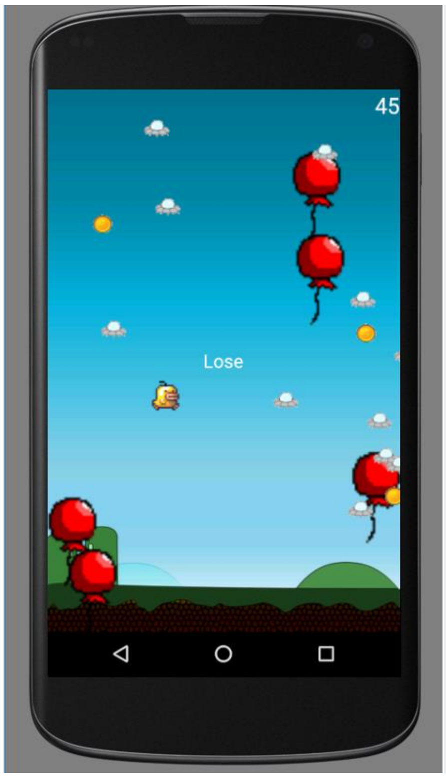
The biggest improvement that I implemented was quite simple but very effective. Instead of using ObjectList.get(i) every time I needed to access the object array when redrawing the images I created a new GameObject which held the ObjectList.get(i) which allowed for access to the array without searching each time. This slight change allowed the game to run slightly, but notably, smoother.

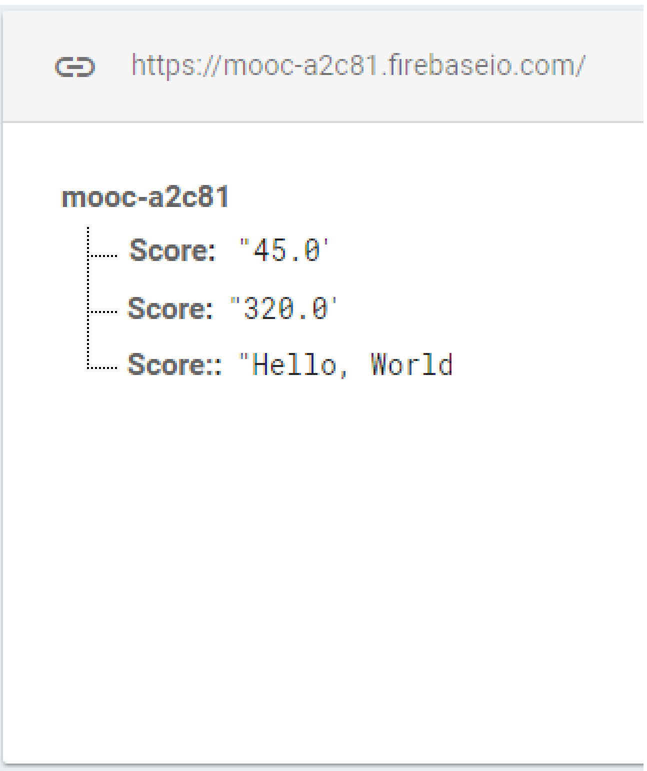
I also attempted to integrate the FireBase database system to allow for player scores to be held online. I managed get the uploading to the FireBase server working which allowed for the score of the current run to be saved, however I ran out of time to implement a reading function that read from the database and wrote out to a high score list. If I had more time and knew about FireBase more in depth I would certainly implement this feature as it allows the users to stay competitive with each other and allow the game to be both about fun and competitiveness.

I tested my FireBase database by simply playing the game. Once the game had ended and the user had finished the game and ran out of lives the score displayed in the top right corner is then uploaded to the FireBase server and stored in a json tree. This can be seen below.

Future improvements that I would like to implement are to get the high scores list working fully, this would incorporate displaying the users score and ordering the scores by highest score. Also, the user interface for the main menu could use an essential makeover to fit in with the game theme. There wasn't a lot about the XML and feature design so I had trouble with creating it.

Result





https://mooc-a2c81.firebaseio.com/

```
mooc-a2c81
├── Score: "45.0'
├── Score: "320.0'
└── Score:: "Hello, World
```

Above is the Game over screen with 45 points, alongside the test database with the latest Score being 45.

Discussion

Changing the ObjectList.get(i) into the abstract object GameObject. This improved performance and also eliminated any crashes that were happening due to null pointer exceptions if the ObjectList.get(i) was being called after the object in position I has been destroyed by a collision method. Also by not getting I each time this increases efficiency as it reduces 1 function call per time this line of code is replaced. Albeit a small improvement when the game is running it can turn into hundreds on function calls saved.

Not strictly improving efficiency but improving user interaction with the game is the high score list that was implemented. It allows for a more social aspect to the game by adding a new feature. By having a high score list, it gives the feeling that the app is more well-rounded and is a finished game from the app store.

Conclusions

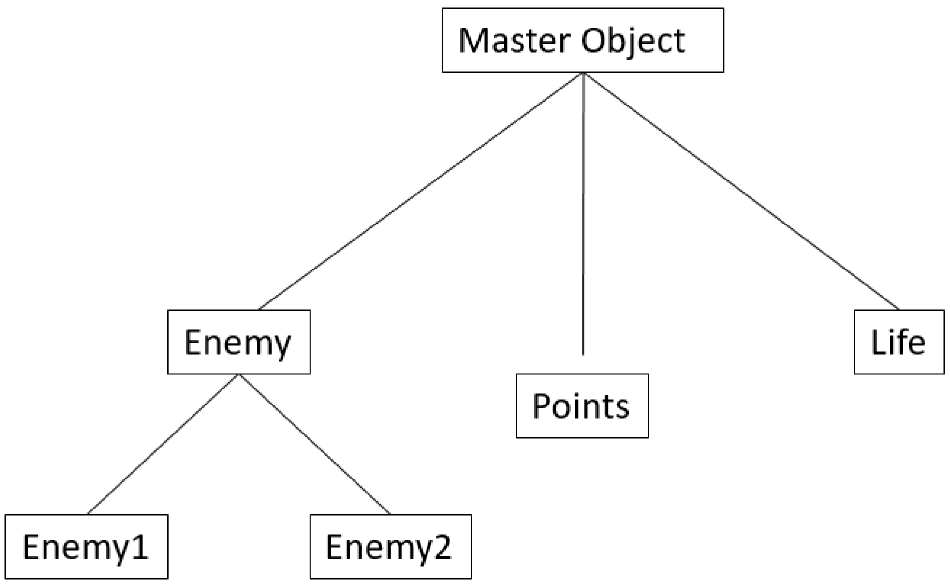
Finally, in conclusion, the online database as the extension task was relatively simple. All that was done is follow the tutorial of FireBase and it's a simple walk though of how to set it up and integrate it with the Android Studio. Apart from the game play it is felt that this is one of the more interesting parts of the project and looking further into FireBase database system would be an interest.

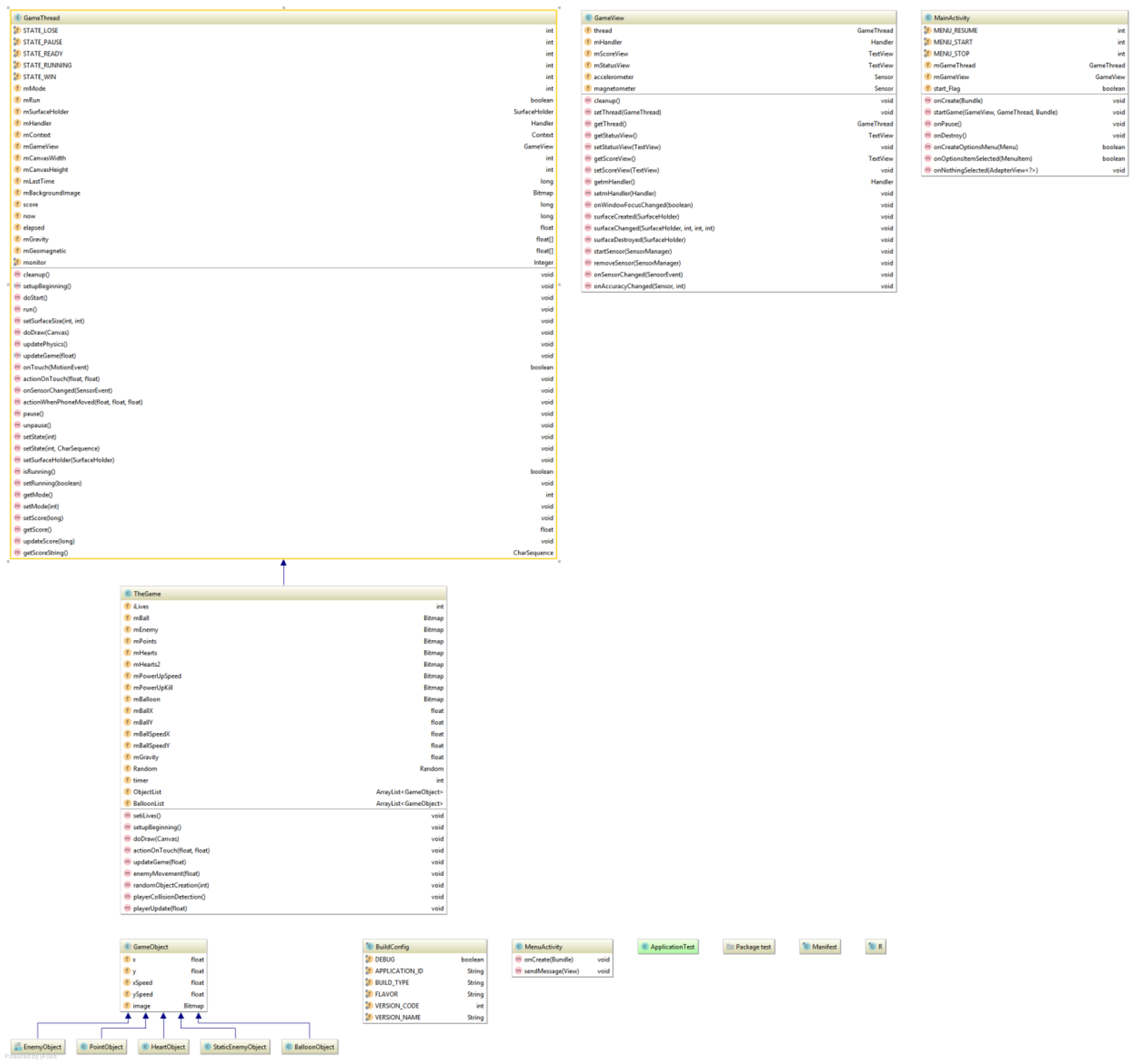
Chapter 5: Final conclusions

To finally sum up this project I feel that it could have gone better, however with the time that was allocated and the resources that were given and available, the best was made. One of the most relevant extension tasks was hit as it enhanced the game experience.

Appendix 1

This appendix should include the original design OOP documentation, any hand written work should be scanned! (Will not be included in page count)




GameThread
STATE_LOSE int
STATE_PAUSE int
STATE_READY int
STATE_RUNNING int
STATE_WIN int
mMode int
mRun boolean
mSurfaceHolder SurfaceHolder
mHandler Handler
mContext Context
mGameView GameView
mCanvasWidth int
mCanvasHeight int
mLastTime long
mBackgroundImage Bitmap
score long
now long
elapsed float
mGravity float[]
mGeomagnetic float[]
monitor Integer
cleanup() void
setupBeginning() void
doStart() void
run() void
setSurfaceSize(int, int) void
doDraw(Canvas) void
updatePhysics() void
updateGame(float) void
onTouch(MotionEvent) boolean
actionOnTouch(float, float) void
onSensorChanged(SensorEvent) void
actionWhenPhoneMoved(float, float, float) void
pause() void
unpause() void
setState(int) void
setState(int, CharSequence) void
setSurfaceHolder(SurfaceHolder) void
isRunning() boolean
setRunning(boolean) void
getMode() int
setMode(int) void
setScore(long) void
getScore() float
updateScore(long) void
getScoreString() CharSequence
GameView
thread GameThread
mHandler Handler
mScoreView TextView
mStatusView TextView
accelerometer Sensor
magnetometer Sensor
cleanup() void
setThread(GameThread) void
getThread() GameThread
getStatusView() TextView
setStatusView(TextView) void
getScoreView() TextView
setScoreView(TextView) void
getmHandler() Handler
setmHandler(Handler) void
onWindowFocusChanged(boolean) void
surfaceCreated(SurfaceHolder) void
surfaceChanged(SurfaceHolder, int, int, int) void
surfaceDestroyed(SurfaceHolder) void
startSensor(SensorManager) void
removeSensor(SensorManager) void
onSensorChanged(SensorEvent) void
onAccuracyChanged(Sensor, int) void
MainActivity
MENU_RESUME int
MENU_START int
MENU_STOP int
mGameThread GameThread
mGameView GameView
start_Flag boolean
onCreate(Bundle) void
startGame(GameView, GameThread, Bundle) void
onPause() void
onDestroy() void
onCreateOptionsMenu(Menu) boolean
onOptionsItemSelected(MenuItem) boolean
onNothingSelected(AdapterView<?>) void
TheGame
iLives int
mBall Bitmap
mEnemy Bitmap
mPoints Bitmap
mHearts Bitmap
mHearts2 Bitmap
mPowerUpSpeed Bitmap
mPowerUpKill Bitmap
mBalloon Bitmap
mBallX float
mBallY float
mBallSpeedX float
mBallSpeedY float
mGravity float
Random Random
timer int
ObjectList ArrayList<GameObject>
BalloonList ArrayList<GameObject>
setiLives() void
setupBeginning() void
doDraw(Canvas) void
actionOnTouch(float, float) void
updateGame(float) void
enemyMovement(float) void
randomObjectCreation(int) void
playerCollisionDetection() void
playerUpdate(float) void
GameObject
x float
y float
xSpeed float
ySpeed float
image Bitmap
EnemyObject
PointObject
HeartObject
StaticEnemyObject
BalloonObject
BuildConfig
DEBUG boolean
APPLICATION_ID String
BUILD_TYPE String
FLAVOR String
VERSION_CODE int
VERSION_NAME String
MenuActivity
onCreate(Bundle) void
sendMessage(View) void
ApplicationTest
Package test
Manifest
R
Powered by yFiles