The Practice of Development "Walk Under Moon" Game

Vladimir Dmitriev, Kirill Kulakov
Petrozavodsk State University
Petrozavodsk, Russia
{vdmitrie, kulakov}@cs.karelia.ru

Abstract

A considerable part of mobile applications market is a variety of 2D games. It includes applications based on original ideas and those that are transformed versions of old classic applications. Our game called “Walk under moon” is also based on one of the classic ideas. The main idea of the game is that the user controls the cat, which runs on the roofs, collect different objects and gets scores for them. This article describes development of this game which is implemented using Qt and Qt Quick.

Index Terms: Qt, Qt Quick, Walk under moon, 2D-game.

I. Introduction

First mobile applications appeared more than a decade ago. At the beginning they implemented the basic functions of mobile phones. Now such applications are complex programs that are designed to help people in their work and everyday life. The market of these applications is growing, new applications are added constantly. It has become so diverse that it is difficult to think up an application that could be urgent. A considerable part of this market is a variety of 2D games. It includes applications based on original ideas and those that are transformed versions of old classic applications. One example of classic old games is series of games called Sonic.

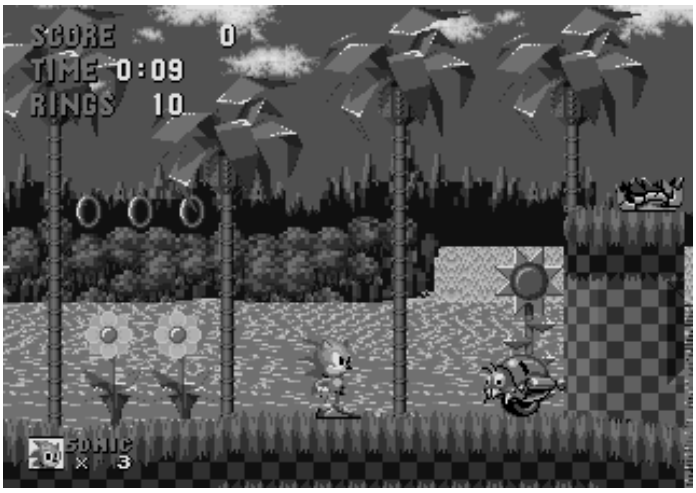

Fig. 1. Screenshot from Sonic game

The first *Sonic* game was a platform game released in 1991 that featured protagonist Sonic running through the game's levels in order to stop Doctor Robotnik from taking

over the world. The game focused Sonic's ability to run and to jump at high speeds with the use of springs, checkpoints, and loops. Screenshot from Sonic game are shown on Fig. 1. See more [1].

Our game is called "Walk under moon" [2], it is based on classical ideas are borrowed from games Sonic and others similar to it. These ideas are that the user controls the hero, who travels in the game world, collecting different objects and overcome obstacles along the way. In our game the main hero is a cat that walking on roofs during a night. To complicate game we made a couple of changes. The first of them is that the cat is always running. And the second one is that the cat cannot kill the enemy, he must to avoid them.

II. DESIGN

At the initial stage of the project the basic concept of the game was developed. It consisted of designing the physics of the game and the level generator.

Originally, we planned to use the Box-2D [3] library as a physics engine. But the only thing we needed was a physics of jump, so we decided to use a simple algorithm, which is described below. Illustration of this algorithm is shown on Fig. 2. We add parameters of velocity along the X and Y Axis to the object. Also we need to use a parameter of gravity. When the object is jumping, it moves along the Y axis with a specified velocity which is gradually reduced by gravity is formally defined as:

$$y(t) = y(t-1) + v(t-1),$$

$$v(t) = v(t-1) - g,$$

where g – gravity constant, t – time value, $t > 0$, v – velocity along the Y axis, $v(0)$ – predetermined constant and $y(0)$ – location of an object on the Y axis before the jump. Velocity along the X axis is constant.

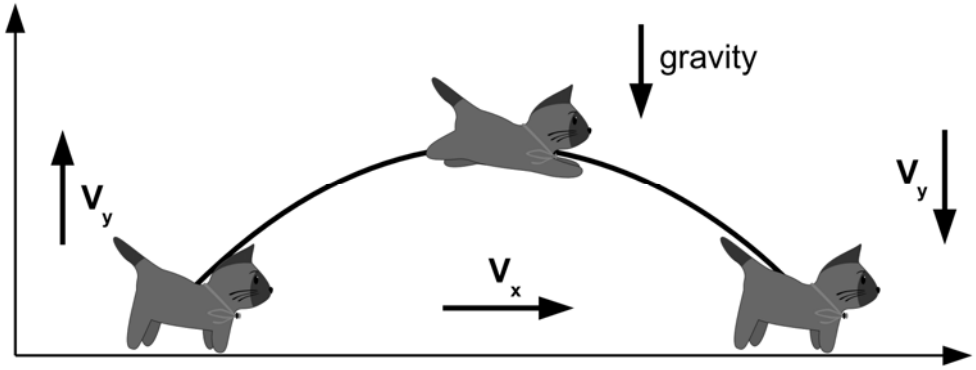

Fig. 2. Physics of jump

Level generator generates all objects, bonuses and enemies. Mainly it chooses their location which is calculated as follows:

$$x_i = x_{i-1} + w_{i-1} + d + e,$$

where x – list of location objects (bonuses or enemies) along the X axis, i – index of object ($i > 0$), w – list of width objects, d – minimum distance between objects, e – extra distance between objects, which is chosen randomly, and $x_0 = d + e$.

Types of the objects, bonuses and enemies are chosen differently.

Type of an object depends on type of the previous one. Each of the objects types belongs to one of the classes, which are grouped by height. Before choosing type of the

next object, we define a class to which the previous object belongs. Then we select class from classes that have lower height than the previous one or higher by one level of height randomly. As a result, type of the object is randomly selected from the resulting class.

Type of a bonus is selected with a certain probability. Occurrence of a bonus, which cost is some scores, has larger probability in comparison with occurrence of those bonuses that affect the characteristics of the cat (for example, additional life).

Type of an enemy is randomly selected, except for those types which are chosen by generator based on the time of their appearance. It can be periodic like for a bird, or it can depend on other parameters of the game. For example, a raccoon appears only when the user gets a certain number of scores.

The next stage of the project was the development of the game design. Usually, the team of developers has a man who is knowledgeable in the design, but we don't have one. Therefore, the process of creating the game design was the most demanding to time resources. You can always find a texture for your game on the relevant web-sites, but we needed our own design, so we worked on every detail of the game by ourselves. The main tool development of design was a vector graphics editor Inkscape [4].

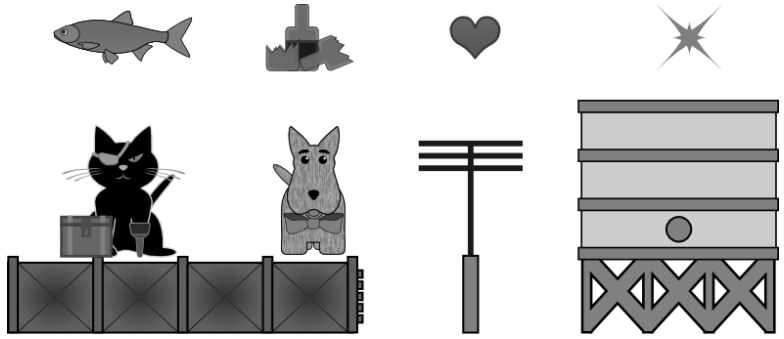

Fig. 3. Textures for game

Because of possible porting of the game to different platforms the game's interface had to be rubber. Therefore, the basic textures for the game were saved in PNG format. Their size had to be fixed. The rest of the textures were stored in SVG format, because images in this format can be scaled without quality loss. Examples of few textures are shown on Fig. 3.

When all the textures were ready, we started to animate the cat, his abilities and enemies. Most part of animation is done using a graphical editor Gimp [5]. Each of the animation layers is drawn in Inskape, and then it is saved as GIF animation in Gimp. Example of animation layers are shown on Fig. 4. The rest of game animations are described in the code by varying the characteristics of the objects.

Fig. 4. Animation layers

III. IMPLEMENTATION

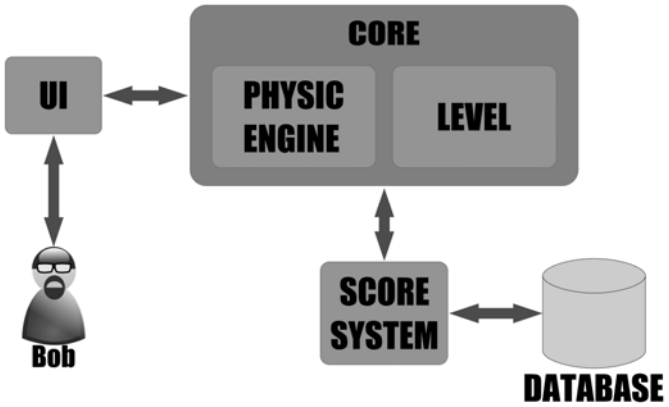

Fig. 5. Architecture

Algorithm 1 Part of generator (placement of objects on the roof)

```
…
component = Qt.createComponent("Object.qml")
…
while (true) {
        dist = getRandom(200, 80) // Randomly chosen distance between objects

        if (i != 0) // If the object is not the first on the roof
           // then take into account the location of the previous object and its width
           x = dist + objectPR[i-1].xObject + objectPR[i-1].wObject;
        else
           x = dist

        // Set the location of object on the roof
        objectPR[i] = component.createObject(pRoof, {"x": x})

        // Choosing object on the roof
        objectPR[i].img = "qrc:/objects/object" + getIdObject()
        // Location on the Y axis set automatically

        // If object is located partially to the roof
        if (objectPR[i].x + objectPR[i].width > pRoof.width) {
           // Removing object
           objectPR[i].destroy()
           delete objectPR[i]
           break // Exit the loop
        }
        …
}
…
```

For the realization we chose Qt and Qt Quick. It allows making creation of the user interface and the logic of the game completely independent stages of development. At these stage we developed the main modules of application. Unlike development user interfaces that implement only one module, the stage of development game logic is divided into development of the following modules: the core application and the scoring system. All of these modules are the basis of architecture, which are shown on Fig. 5.

CORE of the application is responsible for the processing of data coming from the other modules. It includes following modules: **PHYSIC ENGINE** and **LEVEL**. Core and its modules are implemented in JavaScript.

PHYSIC ENGINE is responsible for the behavior of enemies, interaction with objects and the physics of a jump. Each type of enemy has its own behavior, which is characterized by different interactions with the objects and the fact that some enemies can jump.

LEVEL is a level generator. First, it generates the basic characteristics of roofs. Then the generator places objects on them. This part of the generator is shown below in Algorithm 1. Thereafter it places bonuses and enemies. At the end generator starts the animation and behaviors enemies.

```
Rectangle {
    // Path to the image of the object
    property string img: ""
    property string type: ""
    property real size: 1.0
    // Object identifier
    property int indObject: -1

    Image {
        source: img
        scale: size
        smooth: true

        y:-element.height + 1
    }
}
```

UI is user interface, which is implemented with QML. On Fig. 6 you can see screenshots of the menu interface and gameplay of the game. Menu in addition to its basic functions (for example, a new game) allows you to see the table of records, information about the game and in-game store.

SCORE SYSTEM is a subsystem for accessing and working with SQLite databases. It is written on JavaScript.

DATABASE includes files that store records, the characteristics of all cats and game settings, which contain information about selected number of the cat and in-game money.

Fig. 6. Screenshots of UI

The main problem during the test phase of the game was high load on mobile phone. The reason was in dynamic background of game, which implemented such effects as snow, rain, meteoric shower and stars. To reduce the load, we decided to remove the dynamic background from the last version of the game. This decision was accepted because of the project's timeframe. But stars were the part of the basic game background; therefore it was necessary to keep them. To accomplish this goal we recorded video with the stars in the sky from the game screen. Then the video was converted into a GIF animation that became the background of game.

TABLE I
CODE METRICS

Modules	QML files (LOC)	JavaScript files (LOC)
PHYSIC ENGINE	1 (106)	–
LEVEL	–	2 (954)
UI	7 (1920)	–
SCORE SYSTEM	–	1 (436)

Implementation metrics of the application are shown in Table I. It shows the code metrics for the paid version. Differences of the paid version are following: no ads, additional skills for cats. Total number of lines for this version 3416, this is different from the free version on 64 lines.

IV. Conclusion

In this article we described the main development stages of our project, for which implementation we used Qt Quick. We considered code examples from the project and its design. Also we described the main problems and their solutions.

The project was initiated in 2012 at Petrozavodsk State University, FRUCT laboratory

of wireless and mobile technologies. It is published on Nokia Store and has two versions: free [6] and paid [7]. For now the free version has been downloaded 7241 times and paid – 18.

The article was published with financial support from the Strategic Development Program of Petrozavodsk State University.

REFERENCES

[1] Sonic the Hedgehog (series) - Wikipedia, the free encyclopedia. September. 2012. [Online]. Availabe: http://en.wikipedia.org/wiki/Sonic_the_Hedgehog_(series)
[2] Walk under moon. September. 2012. [Online]. Availabe: http://oss.fruct.org/projects/walkundermoon/
[3] Box2D QML plugin – Gitorious . September. 2012. [Online]. Availabe: https://gitorious.org/qml-box2d
[4] Inkscape. Draw Freely. September. 2012. [Online]. Availabe: http://inkscape.org/
[5] GIMP - The GNU Image Manipulation Program. September. 2012. [Online]. Availabe: http://www.gimp.org/
[6] Download Walk Under Moon and many other. September. 2012. [Online]. Availabe: http://store.ovi.com/content/294114
[7] Download Walk Under Moon + and many other. September. 2012. [Online]. Availabe: http://store.ovi.com/content/296146