**CS443 – Final Project Report**

Student name: Xiaoqian Zhang

**1. Project Description**

Project name: Fairy catching – A simple 2D game for android systems.

Goals: Creating a 2D game having following features:

* Player controls main character’s movement (the Hunter) by tapping on the screen. Whenever player taps on the screen, the Hunter will change its movement direction.
* There’re 10 fairies moving randomly on screen. These such fairies and the Hunter will bounce back when they reach to the edge of screen.
* The moving speed of Fairy and of the Hunter can be selected by player when he starts the game. By default, the Hunter will move twice faster than fairies.
* When the Hunter catches a Fairy, this such Fairy will be destroyed. An explosion will occur on screen with specific sound.
* The is always background music playing during the game
* The mission of player is controlling the main character (the Hunter) catching all of fairies fast as possible.

**2. Application design**

*2.1. Scenes of game*

This game has three scenes as follows.

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
|  |  |  |  |  |
| Game options  (scene 1) |  | Main scene  (scene 2) |  | Game Ends  (scene 3) |

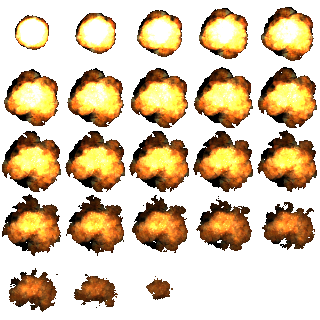
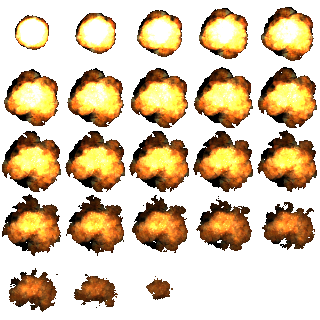
* At the game option, player will declare his name. Choosing the moving speed of Fairy and the Hunter. Player decides to turn background music on/of as he will.
* The main scene will display characters over a background of blue sky. This scene is main part this whole game. There is a game thread taking in charge of displaying characters’ movement on screen. The game uses an inheritance of class “SurfaceView” to draw objects on screen (background, fairies, hunter, explosion)

*2.2. Game objects animations*

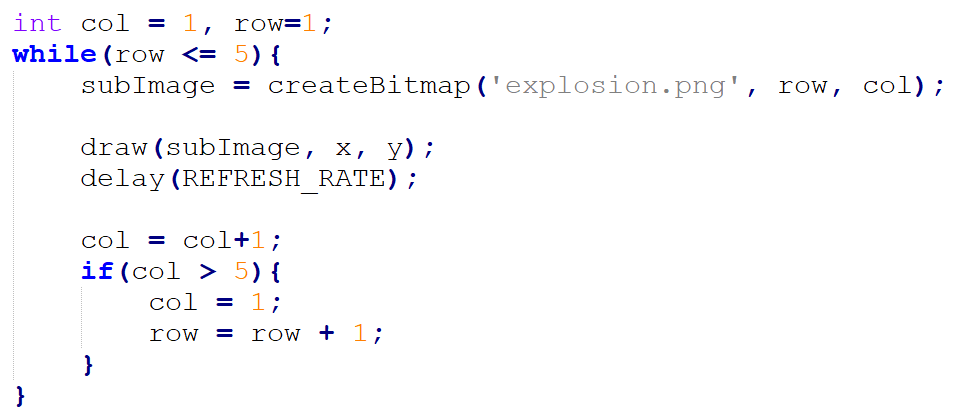
To present movement of character, we use different images appropriate to moving direction. These such images are extracted from a single image file. Each character of the game is stored in an images file containing 4 rows x 3 columns sub-images. Each row corresponding to movement direction:

|  |  |  |
| --- | --- | --- |
| **Hunter’s movement images**  (catcher.png) | **Moving direction** | **Fairy’s movement images**  (fairy.png) |
|  | Goes down  Goes left  Goes right  Goes up |  |

An animation of an explosion will be presented by a series of 25 sub-images storing in single PNG file (explosion.png).



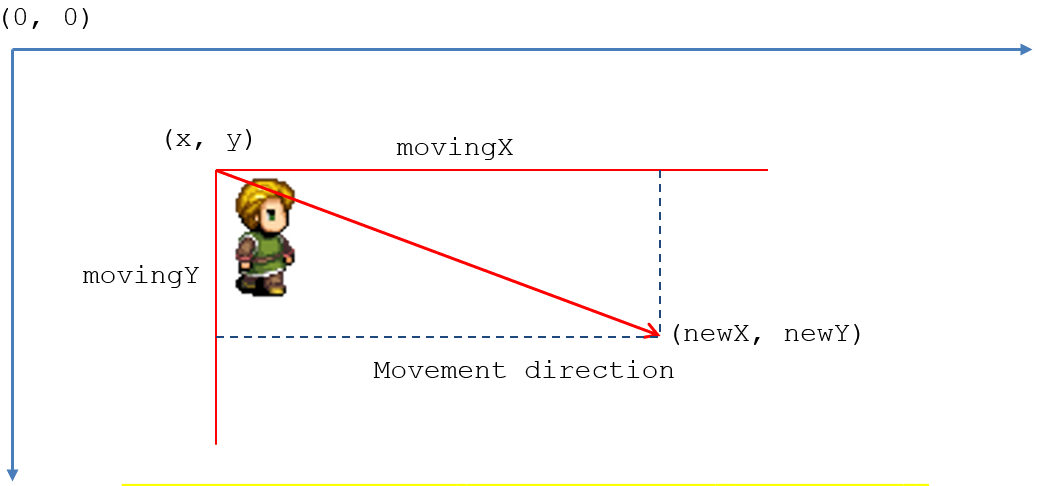
The algorithm showing animation of characters or explosions is simple: We just use an array to store sub-images then draw these images one after one with a delay value stored in REFRESH\_RATE variable (GameThread object). The following pseudo code is applied to get series of sub-mages and display it on a View object.



In the real code, we implemented a class named GameObject containing a method of extracting a subimage from main image file. Then the two classes GameCharacters and Explosion are inherited from GameObject.

*2.3. Characters movement*

To illustrate the movement of a character on the canvas, all we need is redrawing appropriate character’s images at different coordinates. Suppose the character appears at coordinates *(x, y)* at time *t,* and we use a moving vector *(movingX, movingY)* to present the direction of movement. We can derive the new coordinate of character by the equation below:



*distance = velocity \* time\_of\_moving;*

*newX = x + distance\*movingX / movingVector;*

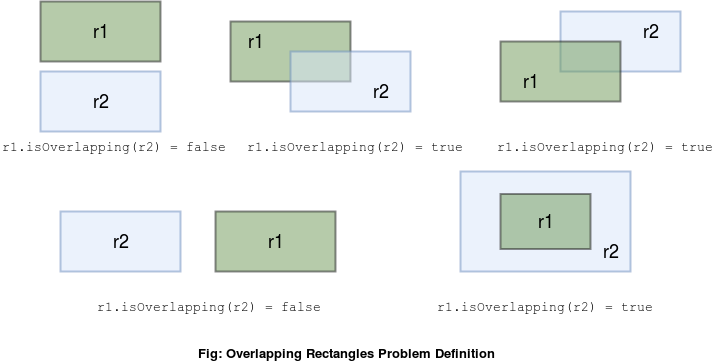
*newY = x + distance\*movingY / movingVector;*

In real code, we the moving vector of the Hunter changes when player taps on screen. The moving vector of a Fairy is generated randomly when a fairy object is created. If new coordinates of character go further edge of screen, then we need to invert value of *movingX,* *movingY* to retrieve a bounce back to center of the canvas.

**if**(**this**.**x** < 0 ) {  
 **this**.**x** = 0;  
 **this**.**movingVectorX** = - **this**.**movingVectorX**;  
} **else if**(**this**.**x** > **this**.**gameSurface**.getWidth()-**width**) {  
 **this**.**x**= **this**.**gameSurface**.getWidth()-**width**;  
 **this**.**movingVectorX** = - **this**.**movingVectorX**;  
}  
**if**(**this**.**y** < 0 ) {  
 **this**.**y** = 0;  
 **this**.**movingVectorY** = - **this**.**movingVectorY**;  
} **else if**(**this**.**y** > **this**.**gameSurface**.getHeight()- **height**) {  
 **this**.**y**= **this**.**gameSurface**.getHeight()- **height**;  
 **this**.**movingVectorY** = - **this**.**movingVectorY** ;  
}

*2.4. Checking two characters is overlapping*

The rule of game indicates that, when the Hunter catch a Fairy, this such Fairy will be killed, and an explosion occurs at coordinates of the Fairy. Since then Hunter and Fairy are presented by two images, so to verify whether Fairy is catch by the Hunter or not, we need to check the overlapping of two rectangles. These following figures demonstrate some cases of the problem:



It is easy to realize that the rectangle *r1* is not overlapped rectangle *r2* when one of these conditions satisfy. Otherwise, r1 is overlapped r2

(r1.topLeft.x > r2.bottomRight.x) OR   
(r1.bottomRight.x < r2.topLeft.x) OR  
(r1.topLeft.y > r2.bottomRight.y) OR   
(r1.bottomRight.y < r2.topLeft.y)

In real code, we implement method *isOVerlapping(GameCharacter other)* in *GameCharacter* class to verify if the current character is overlapped to other character or not.

**private boolean** valueInRange(**int** value, **int** min, **int** max) {  
 **return** (value > min) && (value < max);  
}  
  
*//check if two characters is overlapped***public boolean** isOverlapping(GameCharacter other){  
 **boolean** xOverlap = valueInRange(**this**.**x**, other.**x**, other.**x** + other.**width**) ||  
 valueInRange(other.**x**, **this**.**x**, **this**.**x** + **this**.**width**);  
 **boolean** yOverlap = valueInRange(**this**.**y**, other.**y**, other.**y** + other.**height**) ||  
 valueInRange(other.**y**, **this**.**y**, **this**.**y** + **this**.**height**);  
 **return** xOverlap && yOverlap;  
}

**3. Application implementation**

There’re 5 main classes when we implement this game

|  | **Class name** | **Description** |
| --- | --- | --- |
| 1 | GameObject | This abstract class is father class of three other classes appropriate to different objects of the game (the Hunter, Fairy and Explosion).  A GameObject consists of an image which contains number of subimages for animation of the object.  The most important method of this class is *createSubImageAt(r, c)* which return a sub-image at column *col,* row from the main image |
| 2 | GameCharacter | This class is inherited from GameObject class. For each character (the Hunter or Fairy) of game, we will extract sub-images of each moving direction from main image then store it into four bitmap arrays. *leftToRights*, *rightToLefts*, *topToBottoms* and *bottomToTops*.  The *velocity* variable will affect to moving speed of character on screen. |
| 3 | Explosion | This class is also inhered from GameObject class but the update method doesn’t calculate new position to display sub-images. |
| 4 | GameSurface | This class is inherited from SurfaceView class. This view will display all objects of the game.  In update method, we firstly update new position of every characters. Then we check the position of the Hunter to verify if it overlaps any Fairies, we mark these fairies to be killed, and delete this Fairy from the memory of game.  The constructor of this class generates one Hunter and ten Fairies; draws background of screen and creates an instance of GameThread which will draw the surface periodically |
| 5 | GameThread | This class extended from Thread class. It takes in charge of updating GameSurface periodically. |

Activities

|  | **Activity** | **Description** |
| --- | --- | --- |
| 1 | MainActivity | This activity contains view which is instantiated of GameSurface |
| 2 | GameStartActivity | This activity will display options form for user when the game is started |
| 3 | GameFinishActivity | This activity display the score after the Hunter kills all Fairies on screen |

Resources:

|  | **Recourse** | **Description** |
| --- | --- | --- |
| 1 | Fairy\_ico.png | The icon image for the game |
| 2 | Background.jpg | The blue sky background image for the game |
| 3 | Catcher2.png | Image of the hunter |
| 4. | Fairy.png | Image of fairy |
| 5. | Explosion.jpg | Image describes an explosion |
| 6. | Bgsound.mp3 | Background sound of game |
| 7 | Explosion.wav | The sound of explosion when a fairy is killed |

**4. Future works:**

- The game has not finished showing last scene when player kills all of Fairies. I tried times to create an activity from SurfaceView but the game was not stable. Sometime it work, but sometime it crashes.

- Thi this game, I just trigger and implement simple Touch event. In practical, player can swipe the screen to change moving direction of the Hunter. To implement this, we need to implement some related touch event such as Fling, Drag, Scrol

- The game can be upgraded by allowing player to choose the character as he wants. The movement speed and appearance of Fairies on this game can be more complicated.

- We actually can add some more features to the Hunter (blood level, allies, increasing speed…v.v).

**5. References**

* Android 2D Game Tutorial For Beginner android <https://o7planning.org/en/10521/android-2d-game-tutorial-for-beginners>
* Check If two Rectangles Overlapp in Java <https://www.baeldung.com/java-check-if-two-rectangles-overlap>
* Android SurfaceView Example: <https://examples.javacodegeeks.com/android/core/ui/surfaceview/android-surfaceview-example/>