jArk

Technical Document

Stefano Pongelli Thomas Selber

May 24, 2009

Contents

Introduction

Group Composition

Stefano Pongelli [stefano.pongelli@lu.unisi.ch]

Thomas Selber [thomas.selber@lu.unisi.ch]

The Game

Arkanoid is similar to 'Pong', the player controls the "Vaus", a space vessel that acts as the game's "paddle" which prevents a ball from falling from the playing field, attempting to bounce it against a number of bricks.
The ball hitting a brick causes the brick to disappear. When all the bricks are gone, the player goes to the next level, where another pattern of bricks appear. Each brick could contain a Bonus which gives the player some particular ability, like an Ultra ball, a Rifle on the Vaus, ..

Motivation

We choose to implement Arkanoid because it seemed to offer us a lot of freedom choosing which bonus implement and how, create a level editor, custom levels with strange shapes and so on.
Furthermore we could easily apply the Object Oriented feature of Java, for instance a Bonus is an abstract class with many subclasses, each of which is a specific Bonus (ie. FastBallBonus).
The same goes for the Vaus, for the Ball and for the Bricks.

Requirements

Game Types

In our game we will include two different modalities. The first will be the normal arcade mode in which the player will have to defeat several levels one after the other and make points to place the score into the highscore list.
The second modality will be play a single level, created by the current player or he can choose to play the arcade levels that he has already defeated.
There will also be he possibility to edit or create new levels with a simple and user-friendly Level Editor.

Schedule

Milestone 1: Initial idea

The game should be divided into three main packages.

Model

Classes: Game, Player, Ball, Vaus, Brick, Level, Bonus
Enum: Bonuses, to retrieve bonus with certain probability
Bircks: normal, double hit, triple hit, persistent
Bonuses:

Sticky ball Makes the ball stick to the Vaus

Slow ball Decreases ball speed

Long Vaus Makes the vaus longer

Explosion ball Increases the ball destroying power

Double ball Doubles the balls in game

Rifle Vaus The Vaus wll be able to shoot

Ultra ball Makes the ball ultra powerfull

Double Rifle Vaus The Vaus will get 2 rifles

Cannon Vaus The Vaus will get a cannon

The box A box will surround the game preventing balls from escaping

Fast ball Increases ball speed

Short Vaus Makes teh Vaus shorter

Ghost ball Make the ball transparent and incapable of destroying

False balls Doubles the balls and makes them false

Reset Status Erases all the bonuses

Death Loses one life

Life Increase ones lives

Light off Turn off the light

Controller

Listeners, Utilities classes, Constants of the game, ..

View

Main frame, Game frame, Editor frame, ..

Milestone 2: Ball implementation

- Added inheritance (bonus, brick, vaus now have subclasses for each type)
- Changed packet structure (issue 2 and 3)
- Removed state from vaus (issue 1)
- Solved issue 1 (google code)

Issues:
How to make the ball bounce, solved with current and next position of the ball on the matrix
Bonus returning without passing from ball (to lower the coupling), bonus are managed by game, ball returns the brick to explode and the game class lets the level class handle it and return a bonus.

At this point the project has to be run from bash in order to view the simulation (only the field is shown, the ball is invisible at the moment).

We have few tests, but the test we have comprehend a lot of methods.
Moreover this is an early stage of our project, we have to improve a lot of things and change a lot more.
The most important element at this time, is the ball. And this is tested.

Milestone 3: Basic GUI

For this Milestone we implemented a basic GUI with mouse and key event listener.
A main Thread provides us with a sort of "tick" where we repaint the game and update everything.

Issue:
How to compute the intersection between the ball (which is not a pixel anymore)

and the bricks
How to make the ball bounce on the vaus
How to structure the view (divided in panels, ...)

Milestone 4: Bonus and brick

Solved Issues:
The view has to know the Model, and not the inverse.
We had to refactor the project. At this moment Model and View are completely divided.

The different images of each element (bonus, vaus, ball and brick) were inside the element, and each time they were needed they were created (new).
Now the images are all in a class in View and there is only one instance of each image for all the elements (they are created wih the constructor of that class and only the reference is passed on).
We had to add Listeners in order to apply the changes given by bonuses (ie. to the vaus)

Implemented:
Bonuses: everything except for shooting vauses and sticky ball (lives are half implemented)

Brick types:
We have 4 different bricks:
default, destroyed the fist time it is hit;
persistent, which doesn't get destroyed;
resistent, needs to be hit 2 times
very resistent, three times.

Images:
Each bonus/vaus/ball/brick has its image.
Those images are created inside an hashmap in viev, ImagesReference, and only a reference is given to the GamePanel which draws them.

We changed to Swing Timer instead of Threads.

Furthermore, we added more tests, fixed a lot of bugs, and refactored the entire project.

Milestone 5: Level editor

Done:
- lots of bugfixes.
- Level Editor (with save/load/test capabilities).
- refactoring (listeners).
- GUI (general).
- implemented all bonuses.
- implemented bonus life span in the game frame.
- implemented general info (lives, points, ..).
- game over/next level (random at the moment)/replace ball after loosing life.

Problems encountered (solved):
- find a "good" way to implement the editor.
- display taken bonuses and their duration on main screen.

Problems:
- code duplication (ie. bonus strings).

Milestone 6: Project complete

We made a lot of changes in this Milestone:
instead of loading all the images at the beginning we now use a sort of "lazy loading": whenever an image is required it gets also stored in a Hashmap, if the same image is required another time, it gets taken from the HashMap directly. Furthermore we use Properties file sto store the path to the images and default levels.
The GUI has been finished, with a perfectly working level editor and a main frame with a card pane to display the different things.
We implemented the HighScore, a new Bonus (turn off the light) and much more.

The only thing we miss at this point, is the sounds.
But we had some trouble implementing them (not working properly), so we decided to not have them.

The project can be considered complete.

Design

Game

This is the class that runs and coordinates the core of the game, it holds references to all moving and non-moving objects in the playing area. Here we have a tick method that steps the whole world forward and some method to check if the game is over or if the level is cleared.

Level and LevelManager

This two classes are here to assure the correct storage, load and creation of level. The Level class also provides methods to destroy brick and retrieve the bonuses that are in there. The LevelManager class centers its functionality providing methods to load and store level into files, mainly used to connect the Level Editor with the Game and the Game Frame.

Ball

The abstarct class Ball with its subclasses holds all the necessary information about position, speed and how to move a specific ball object. A Ball is modelized as a square and, to let it bounce correctly, at every tick it checks the four corners, it begins by checking vertical and horizontal axis and then it looks at the diagonal, in case that one of the checks returns true, tha ball will be bounced on the opposite direction and destroys the brick.

Bonus

Bonus is an abstract class that is extended by 3 different types of bonuses: PlayerBonus, BallBonus or VausBonus. Each of our 18 bonuses extends one of those three bonus type. A bonus applies itself to a game with the method apply(Game game) and removes the effect with the similar method remove(Game game). Every bonus has also a bonusClass, an int used to identify different classes of bonus of the same type, so that we can overwrite bonus of the same type and bonusClass.

Vaus

Another abstract class, here we store all we need to know about the Vaus, with the 4 subclasses it's possible to have all the different paddle that we need and move them.

GamePanel

This class provides us an object able to translate what happens in a Game object to something visible and allows the user to interact with the core of our game.

References

General Informations

We took all the information we needed from:

- **http://en.wikipedia.org/wiki/Arkanoid**
  Gener overwiev of the game and short description of the original version.