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

Much like the game '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 striking 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. There are a number of variations (bricks that have to be hit multiple times, flying enemy ships, etc.) and power-up capsules to enhance the Vaus (expand the Vaus, multiply the number of balls, equip a laser cannon, break directly to the next level, etc), but the gameplay remains the same.

Motivation

We chose to implement Arkanoid...

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.

Milestones

M1, initial idea

/* MODEL */

// class to describe ball
class Ball
// class to describe the vaus/paddle
class Vaus
// class to describe a brick
class Brick

birck types :
- 0 : normal
- 1 : double
- 2 : triple
- 3 : fixed

// class that describes the position of the bricks on a level (maybe parsing files)
class Level
// class that contain information about the player
class Player
// class that holds information about bonus
class Bonus:
- 0 : sticky ball(1)
- 1 : slow ball(1)
- 2 : long vaus(1)
- 3 : explosion ball(2)
- 4 : doubling ball(2)
- 5 : laser(2)
- 6 : ultra ball(3)
- 7 : double laser(3)
- 8 : missile (4)
- 9 : the black box (5)

- 10 : fast ball(1)
- 11 : short vaus (1)
- 12 : ghost ball (2)
- 13 : block freeze (inefficient ball) (3)
- 14 : reset vaus (3)
- 15 : false balls (become true after time) (4)
- 16 : death (5)

// enum to hold the different types of bonus
enum BonusEnum
// class that holds information about the current state of the game
class Game

/* CONTROLLER */

// class to parse mouse/keyboard inputs
class InputParser
// enum with messages (maybe taken from a file?)
enum MessageEnum
// class to receive input and print outputs
class Console

/* VIEW */

// class to design the output (draw all)
class Field

M2: Ball implementation

- Added inheritance (bonus, brick, vaus)
- Changed packet structure (issue 2 and 3)
- Removed state from vaus (issue 1)
- Solved issue 1 (google code).

- issue: bouncing method.
- solved with current and next position of the ball on the matrix.

- issue: bonus returning w/o passing from ball (to lower the coupling)
- bonus treated by game, ball returns the brick to explode and the game class lets the level class handle it and return a bonus.

- 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)
to run it: just press enter continuously, or type something to quit. You will see

the bricks hit by the ball disappearing.

****
Pay attention: also if we have few tests, we are quit sure the implemented things are working properly.
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.
****

M3: Basic GUI

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

Issues: 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, ...)

M4: Bonus and brick

Issues: .jar not working

TODO: we have to finish implementing some bonus
add some listener
add player support
add ending event
finish the gui
add a level editor
add an algorithm to create levels
...

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 view.ImagesReference, and only a reference is given to the GamePanel which draws them.

Swing Timer instead of Threads

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

M5: 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).

M6:

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 funcionality providing methods to load and store level into files.

Ball

The Ball class 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.

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

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.