3D Modelling & Animation

Coursework

Alexander Zierbeck
MSc Artificial Intelligence
091605252

Markus Nehfischer
MSc Artificial Intelligence
091605218

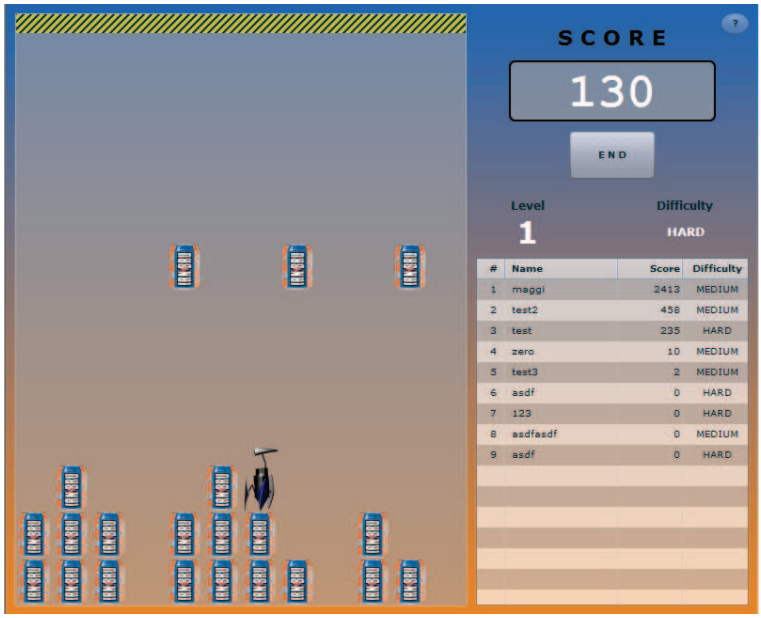
04. December 2009

Part 1

Executive Summary

In the course of an inoffical, probably illegal and not really existing partnership with A.G. Barr plc the well known producer of the famous and very popular Softdrink IRN BRU© we are proud to present the result of our months of efforts :

STACK-IT



The game excells in the genre of fast paced (almost) action and puzzle games and thrills the player with the omnipresent danger of IRN BRU© cans falling on his head or get stuck inbetween towers of IRN BRU© cans and get burried under more and more IRN BRU© falling from the sky (seems to be not that bad death for me...).

While escaping the threat of beeing smashed by some solid IRN BRU© cans the player can move cans that reached the ground by controlling some cute little robot and make them disappear by completing the ground row. For each deleted row the player scores points which get saved in the high score. And if the fun playing STACK-IT weren't enough the best player in the high score table at the end of each month gets free IRN BRU© for the whole next month! (no guarantee for this yet)

Game objectives and play

STACK-IT is a single-player Puzzle-game in tradition of Tetris (e.g. on Nintendo's Gameboy) or STACK-ATTACK (on former Siemens mobile phones). Goal of the game is to score as many points as possible and climb up the high score ladder. Points can be scored by sorting falling cans to a full ground row. Every time the ground row gets completed it will be deleted and the player scores points depending on chosen difficulty and number of row deleted.
To move the cans to the desired place the player can control one of two different robots.

Controls:

KEYS	ACTION
Left	Move the robot to the left.
Right	Move the robot to the right.
Up	Lets the robot jump (one row in height)

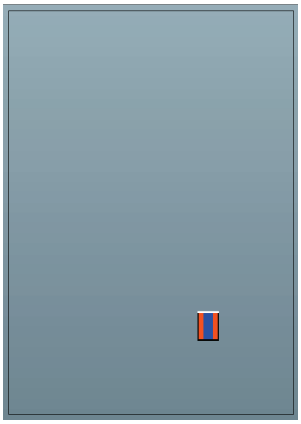
Every time a can loses contact to its ground (e.g. row below gets deleted or robot pushes the can over the edge of the row beneath) the can starts again to fall until it reaches ground again.
The robot also reacts to this kind of physics and will fall if he loses contact to ground.

Hint: Try to avoid falling in holes with both neighbouring columns being at least 2 can heights higher. You won't be able to get out of there because the robot can only jump one can height (= row) at once.

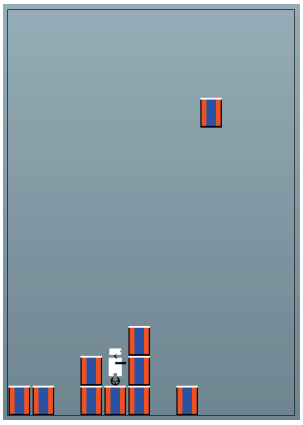
Game over:
The game ends when a can falls on the robots head or if the stacked cans reach a predefined height. In this case the game stops to react to user controls and shows a popup where the user can enter his name. Afterwards the points the player scored get added to the high score table.

Game Development

We decided to build a game similar to one on old Siemens mobiles. It was called STACK-ATTACK and the objective was similar to Tetris: complete rows to delete them and score points. The first drafts included the idea of a box/can distributor in the upper right corner which throws the boxes in the playfield where they bounce from the walls until it reaches ground or a specific height. But we realised it will bring a lot of problems (no straight fall, no defined columns, time effort, …) so we skipped this part and concentrated on the actual game. But the thoughts on this showed us some parts we would necessarily need.



The first prototype

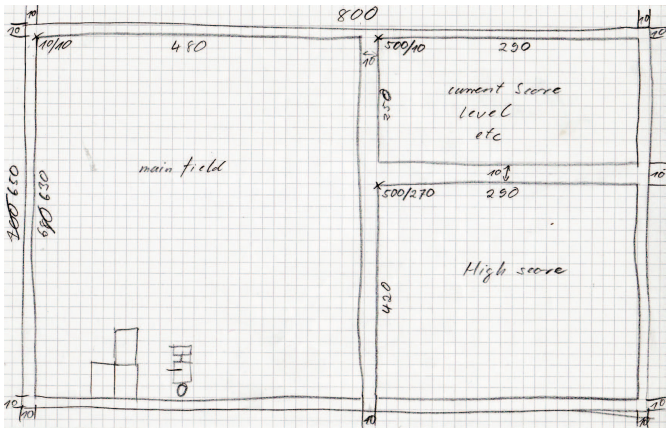


Another prototype in the middle of development

In the first drafts the layout was already divided in three parts. Covering the whole left side and the middle of the application there should be the playfield. On the right side the score had is place in the lower corner and the top corner there ought to be the distributor of the cans. After we decided to skip the distributor the top corner was the new home of the score making space in the lower part available for the high score table. This decision made one of the robots jobless so we implemented an opportunity for the player to choose between two different robots. As a further requirement we discovered the need of some easy way to manage the positions of the falling and the already fallen cans. First idea was to allow complete free movement. But this way had similar problems like the ones before. For example, how to determine when a row is full if it is possible to let space between cans that cannot be filled by other falling cans? So we decided to

use a system of rows and columns to sort in the cans. A common approach to realise such structures is the use of 2D-Arrays. In our case the first dimension represents the columns and the second the rows.

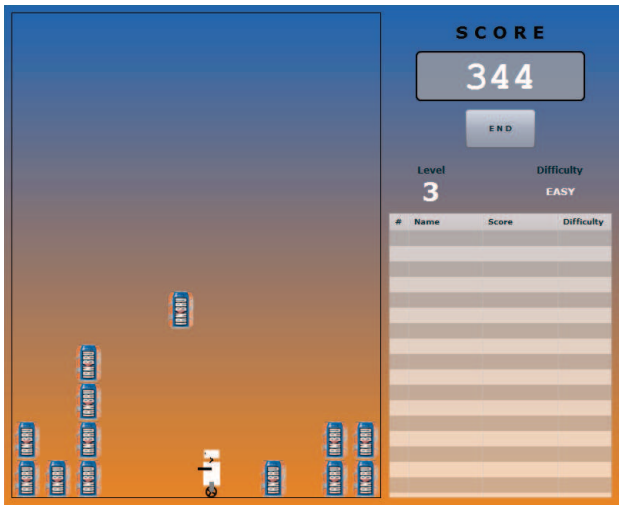
For the animations, e.g. the movement of the robot, we needed some methods to be called repeatedly. In a first attempt we used a timer, but run into trouble when trying to synchronise it with the rest of the application. Our solution is the use of another method which listens to OnEnterFrame events and which is additionally some kind of "synchronised" with the main-repaint – method by the use of a Boolean variable. Not very elegant but it works.



One of the skteches to discuss the overall layout (advanced state of development as can be seen at the right side)

Development steps

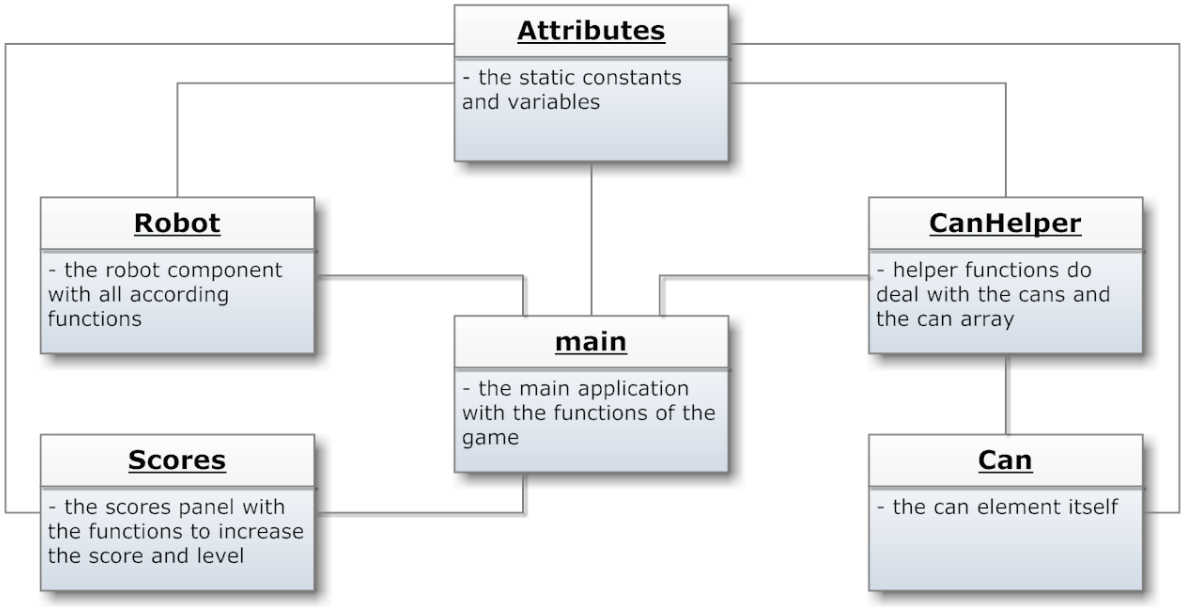
The code development was an iterative process. As the gravity (falling cans and robot) would be one major aspect of this game the first prototype was a single falling can. In the next one the multiple generation at random positions was implemented. In this phase we stumbled across the free vs. column bound position problem mentioned above. The next step was the implementation of the robot and his ability to move to the left or right and jump. One iteration further the robot learned how to push cans and which cans can be pushed (only the ones with no other cans next to it). After this step the game mechanics were completed and we had could test the game on this prototype for the first time. In the following we implemented the score system followed by the database based high score table (using Mike Chantler's php scripts). Afterwards the opportunity to choose the difficulty and which robot model should be used was integrated. And in the closing step the final images of the robots and cans were implemented and the whole application got its colours.



Application in almost final state

Overall Design

- The whole program separated into six components. The main component of the game is the main class. In there are all the functions of the main field and it controls most other components.
- The next important element is the Robot component. This is basically the robot, which is controlled by the user. All images of the robots (both robots) are included in there as well as the functions to repainting and movement. The main component creates a new Robot element in the init function.
- The next component is the Can. This is basically only the image of the can with the
- functions to move and collision detection. The Can is only used by the CanHelper.



- The CanHelper is a component to control an array of Cans. It contains two arrays, one for the falling cans and one for the cans that already reached the ground. Why we did this is described in Part 3. The CanHelper also contains the functions to manipulate the arrays (slicing, pushing, ..) and to create new cans. This component is used by the main.
- The scores is only a visible element to bring the informations, that are calculated in the beckend to the frontend to show them to the user. Furthermore, a few popups are defined (for selecting the robot and the difficulty, the game over dialogue, the help section). This element is created by the main and gets displayed on the right side of the application.
- The Attributes is a collection of all important elements (variables and constants) which are used by more than one component. This makes it easier to manipulate or get the values of a specific attribute. It not linked to a component anymore.

Joint Conclusions

It is really hard to tell which part of the game is the best, because the game only works as a whole. Apart from playing it the development part was also fun. At least some times. But if we really have to pick one part it would be the 'physics system' which causes can and robots to fall if they lose contact to the ground.

However there is always a way to improve things. In the case of STACK-IT there are due to the limited time some more ways. :)

Number 1 on our wish list is to implement some kind of power ups / pick ups. This can be some objects falling down like the cans but if caught by the player it gives the robot some special abilities like more speed, slower spawn rate of the cans and therefore more time to react for the player. Also the ability to jump up more than one row or to delete any row or any can could be possible.

Also we would like to give different behaviours to our robots. For example to let the robot with legs jump one or two rows higher but give him a slower movement.

Next point could be the implementation also more and better graphical effects like smoother animations (especially jumping) exploding cans, flashing power ups, ...

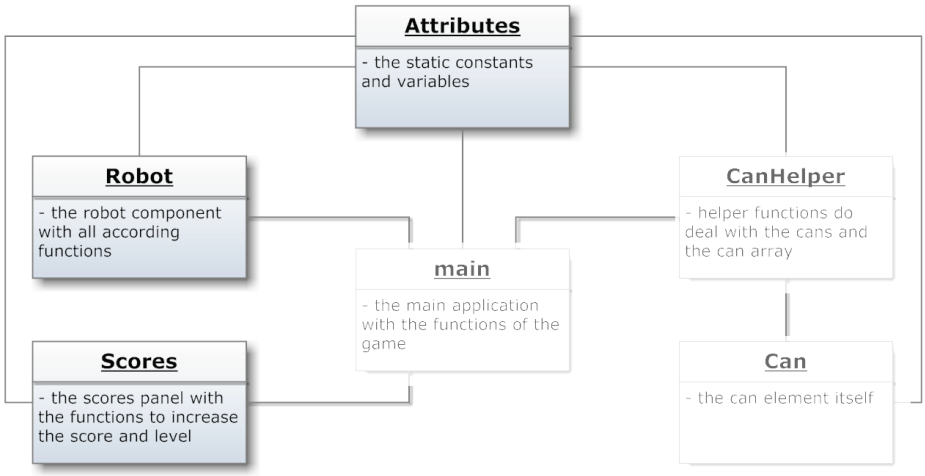
Also the implementation of advanced physics would be nice. For example as planed in the very beginning another bot throwing in the cans from the side and letting the cans bounce in the upper part of the screen before they get 'assigned' to their column.

And at last a function to pause the game would be probably appreciated by players as well as the opportunity to configure the controls by themselves.

Part 2

Summary of Alexander Zierbeck's Role

The packages I worked most with were: **Robot**, **Scores** and **Attributes**.



- My main part was to implement the robots into the game. This class contain the images and the functions to control the robots. This component is essential for the game and is closely linked to the main. The code I wrote for this component is almost 100% own code. Only the repeater part (in mxml) was inspired by the code examples form Mike Chantler.
- The Scores component is mostly to visualise the informations to the user. Its located on the right of the game. This component is 100% own code.
- The Attributes component is a ActionScript class to gather all common used variables and constants. All elements in this class are static to provide the access. This is a simple way for us to deal with the attributes of the game (like size of the can, gravity, …). This component is 100% own code.

Module descriptions

Robot

First of all I'll explain the function of the class Robot. This mxml component is essential for the gameplay because it contains all the methods to control the robot. It also contains the images of both robots. The methods work equal for both robots. Just the images are changed.

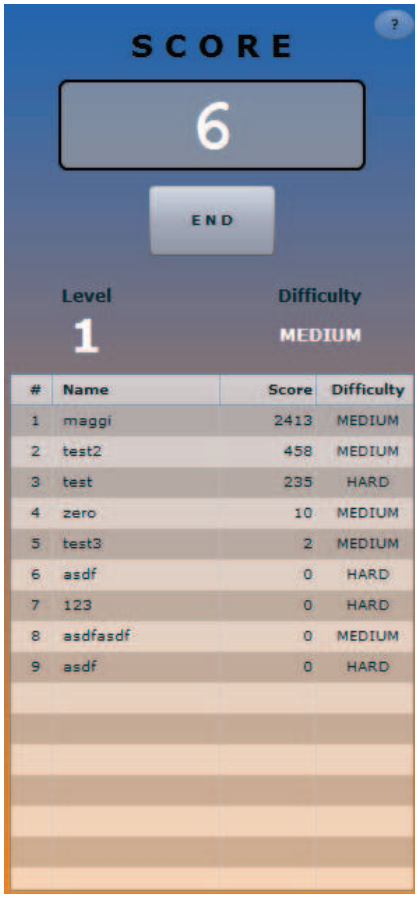
The images are included with the Repeater element from Flex. The number of images is always three. This is enough to get the animation we need for the movement.
Because of the two dimensional view, we only need the three images from the left and three from the right. The final images look like this:

In the development we used very simple example images to get an idea of how the character works.

The Robot has besides the initial function (set location, initialise visible images, …) and the repaint (responsible to show the movement and switching the images of the robot) function the following methods:

- **Falling**: add gravity to the robot, that it falls when a row gets deleted. This is provided by the function *fall*();
- **Move the robot**: this are two functions: the *moverobot*() gets called whenever a right or left key gets hit. It asserts a new value to the column attribute of the robot. The *moveBot*() is to show the movement in the application. It gets called on every onEnterFrame event.
- **Jumping**: the function *jump*() this sets a new height for the robot. Together with the gravity (the *fall*() function) it looks like, that the robot jumps.

Scores



The scores class is the panel on the right side. It shows the current score, level, and difficulty. Also included is the highscore table. Furthermore a start button is added to start the game and choose the robot and the difficulty.

The panel contains a lot of GUI elements to bring the information up to the front.

The highscore field is a component, which connects to a mySQL database. This is provided by a php file (the scoreHandler.php). The xml stream, that is returned by the database, is directly written into the DataGrid.

The Scores also contain three different popups. This are TitleWindows combined with the popup manager. This makes the background unreachable when the popup is opened.
The popups are:

- **Choose robot and difficulty**: the function *showDiffDialog*() opens a window where the user can choose the robot and the difficulty. After the selection of the difficulty, the game starts. The function *diffButtonPressed*() sets the difficulty (which the user selected) and closes the popup.
- **Game over dialog**: when the game is over, a popup will show up (*showGameOverDialog*()). In there is a TextField where you can enter your name to submit your points to the highscore. The submit button will transmit the data to the database (*submitButtonPressed*()). If you do not want to do this, you can close the dialog without submitting by clicking on the cross on the upper right (*gameOverWinClose*()).

- **Help dialog**: The button in the upper right of the application (the question mark) will lead to the help section (*showHelp*()). This shows a simple dialog window with info and advices for the user. Closing again by clicking the cross (*helpWinClose*()).

This component also contains the score and the level information.
These attributes are manipulated by the functions:

- *increaseScore*()
- *increaseLevel*()

The devolopment of this component was not so hard and pretty straight forward. I just added each element one by one and connected the elements with the “backend”.
There is not very much calculation or logic in it. Its just showing information to the user or asks for input.

Attributes
The Attributes component is just one class where all constants and variables are defined, which are used by all other components. There are no functions in this class.
The advantage is that in every class the attributes can be accessed by typing ATTRIBUTES.attribute but they have to be static and public. This also brings a lot more clarity into the program and makes it easier to handle global values.

I developed this class, because we defined variables in each other component and we need to change or read this value in other components as well. The result before the Attributes component was very obscure. There were links from everywhere to everywhere. So we decided to create this component.

What I have gained from this project

First of all, the whole flex and mxml stuff is new to me. I have a experiences in Java / JavaScript and xml, so the development was not a huge problem. The mxml is pretty easy to understand if you know xml. The development in the Flex Developer was also pretty easy (with the design mode and the code completion).
I only had some trouble with the ActionScript. Because that it is basically JavaScript, it has some strange behaviours sometimes. In some cases I was searching for ages for a solution or a way around.
The Adobe Flex Language Reference is a very good site to get all informations to the flex elements and attributes. This helps a lot in the development process.

What I gained from this project is the new view on creating internet games. I haven't done anything with flash jet but I think with flex it is a easy thing to do and it was really fun to create this game.

Appendices

Robot

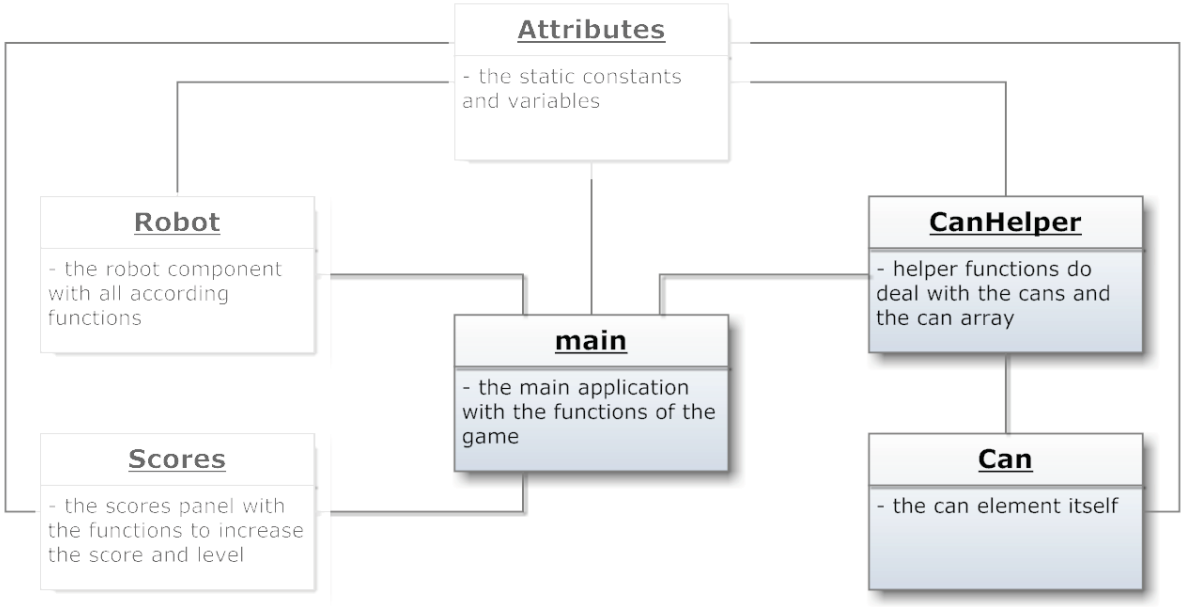
```
<?xml version="1.0" encoding="utf-8"?>

<!-- ====================================================================
* Author:               Alexander Zierbeck
* Date:                       04.12.2009
* References:           -
* Function:             the robot component
* Usage:                just make a new instance of this object.
                              - fall(): let the robot fall with a specific
                                   gravity
                              - moverobot(): changes the column attribute of
                                   the robot
                              - moveBot(): moves the robot on the canvas
                              - jump(): lets the robot jump
* code author:          100%
```

Part 3

Summary of Markus Nehfischer's Role

My role in this project was to develop the main class and the creation and management of the cans. The first part represents the entry point of the application and controls overall behaviour of it. The Can module represents the look and behaviour of the cans and the CanHelper module is responsible for the management of all the cans.
Planing and management of the project was done by both of us in equal parts.



All modules developed by me (main, Can, CanHelper) contain 100% own code.

Module descriptions

main

This module is realised as mxml file entry point and application control. At first the application gets initialised with all necessary values. After this the repaint() method takes over control handling the application from there and update it with every OnEnterFrame-Event.
The control of the can generation is handled by a Timer. A timerCompleteEvent creates new cans by creating objects of the module "Can".

Methods:
init(): initialises the application
keydownEventhandler(): handling input from user
repaint(): redraw the scene and act as main method to control the application
killGroundRow(): delete the ground row (if it is completed) and call functions to deal with score and level.
MainTimerComplete(): add new generated cans to application
crashTest(): do hit test with robot and cans above it
gameOver(): stop the animations and block key inputs

Most "valuable" or important part is the repaint which essentially controls the whole application.

Can
Also a mxml file. It represents the can itself and is instantiated many times.

Methods:
init(): act as constructor and initialise height and width of a can
fall(): let the can fall depending on the used gravity
moveCan(): move the can on the x-axis (when pushed by the robot)
reachedGround(): determine if the can already has reached the ground (level ground or top of another can)

CanHelper
This module is pure ActionScript and controls and manages the two arrays containing all the cans. One array contains all cans which are falling at the moment. The other array, a 2D array, contains all the landed and stacked cans. This partitioning is used to make hit tests easier and less performance intensive because you don't have to check with all cans.

Methods:
initArr(): initialise the 2D array which hold the landed cans
createCan(): create new cans an put them in the array for falling cans
switchArray(): move a can from the array of falling cans to the main array which holds the landed cans
pushCan(): move can in the main array from one column to another

What I have gained from this project

I gained the first practical knowledge in Flash, Flex, ActionScript and mxml from doing this project. While I was already used to Java development it was a new and interesting view on this technologies. Not few times I cursed at things I am used to in Java and not working in Flex, but on the other side it is an interesting way of putting together applications like games or other animated things. One big advantage of Flex is, that the applications can be run using a browser with a Flash Player installed. Today nearly every PC fulfils this requirements (you need Flash to view clips on youtube for example) so the applications you build are runnable on almost every machine. Here and there we stumbled over some strange behaviour which often could be traced back to security issues with flash. So with this project I was able to broaden my horizon in the field of developing applications using various technologies.

Appendices

main

```
<?xml version="1.0" encoding="utf-8"?>

<!-- ====================================================================
* Author:               Markus Nehfischer
* Date:                       04.12.2009
* References:           -
* Function:             the main application of the game
* Usage:                just make a new instance of this object.
                              - keyDownEventHandler(): checks what key is
pressed
                                    and processes the input
                              - crashTest(): checks if a can hits the robot
                              - gameOver(): gets called when the crashTest is
                                    positive
```