Simple, Interactive, Intelligent, Innovative

Introducing Code Rally

An interactive racing game built using IBM tools and runtime

Agenda

- ❑ Motivation for the Code Rally Game
- ❑ Playing the Code Rally Game
- ❑ Development of the Code Rally Game

Motivation for the Code Rally Game

Introduction to Code Rally:

- Code Rally is IBM's latest game focused on the developer community
- Code Rally is first and foremost a fun, social racing game
- Available now for gamers
- Developers can modify features in the source code of the game

Code Rally Objectives:

- Code Rally is meant to increase awareness of the IBM developer tools and middleware using an interactive multi-player game
- Code Rally is hosted on the IBM SmartCloud, supported by a Community Site on developerWorks and development is managed on the open JazzHub platform in an open, interactive environment

Motivation for the Code Rally Game

- The team wanted a game that could be played with one person or with multiple people
- Players select and/or design vehicles that race through a chosen track with obstacles towards a finish line
- Vehicles implement an event driven model where specific events call methods within the “AI” implementation
- Players submit their vehicles to a game server and play occurs automatically, resulting in a video of the action, game statistics, and a view of racer code that is executing
- The game ends once all racers cross the finish line or if the race is not finished within a reasonable time (5 minutes)

Ways to play the game

The user can play the game in many different ways:

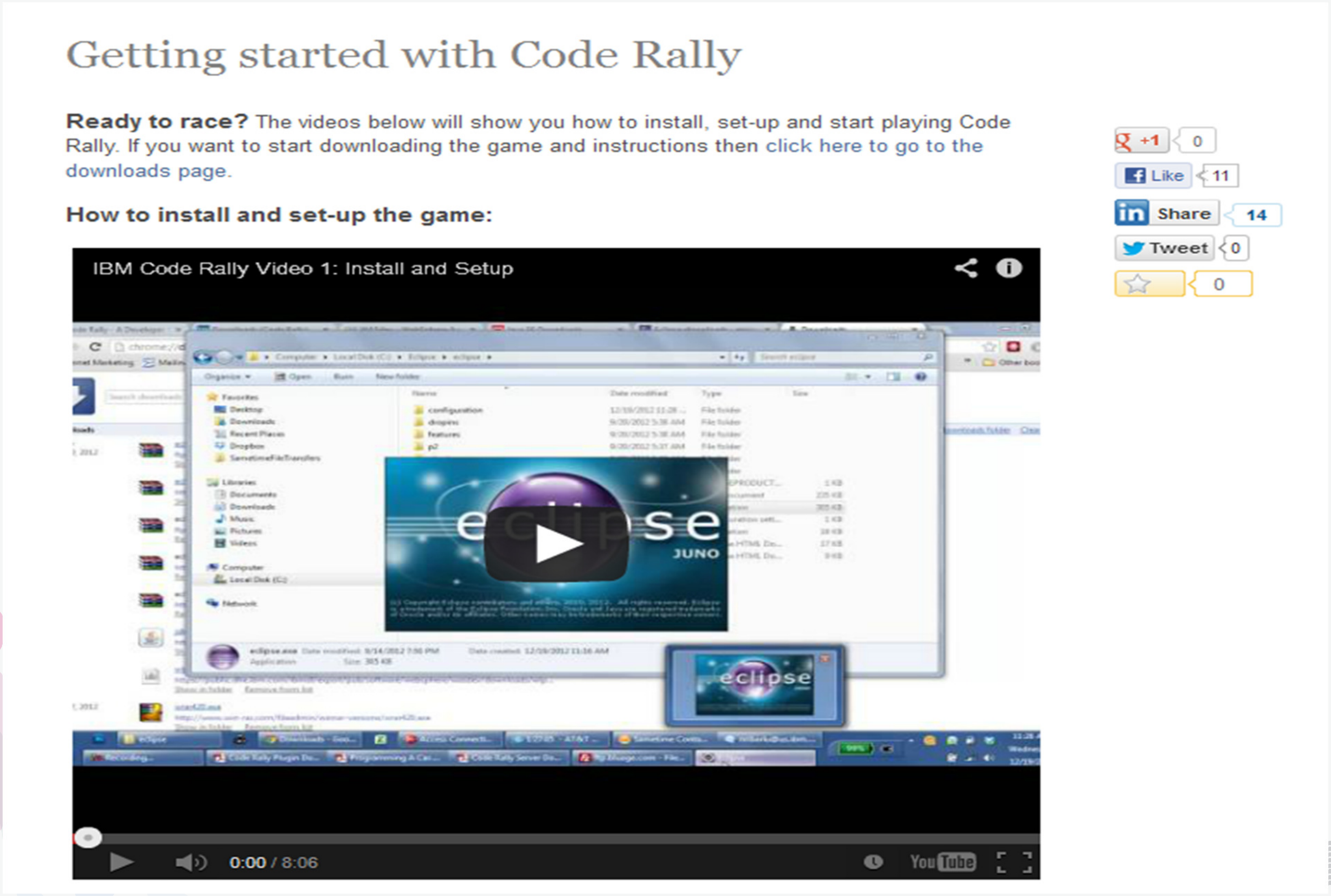
1. Download the tech preview mobile client (Android) only and play on the IBM Smart Cloud

2. Download the java client and play on the IBM Smart Cloud

3. Download the java client and runtime and play locally

Where to get the game?

https://www.ibm.com/developerworks/mydeveloperworks/blogs/code-rally/?lang=en



Getting started with Code Rally

Ready to race? The videos below will show you how to install, set-up and start playing Code Rally. If you want to start downloading the game and instructions then click here to go to the downloads page.

How to install and set-up the game:

Let’s play the game

Developing the game

The Code Rally game was managed using:

- Jazz hub (https://**hub.jazz**.net/)

The Code Rally game was built using:

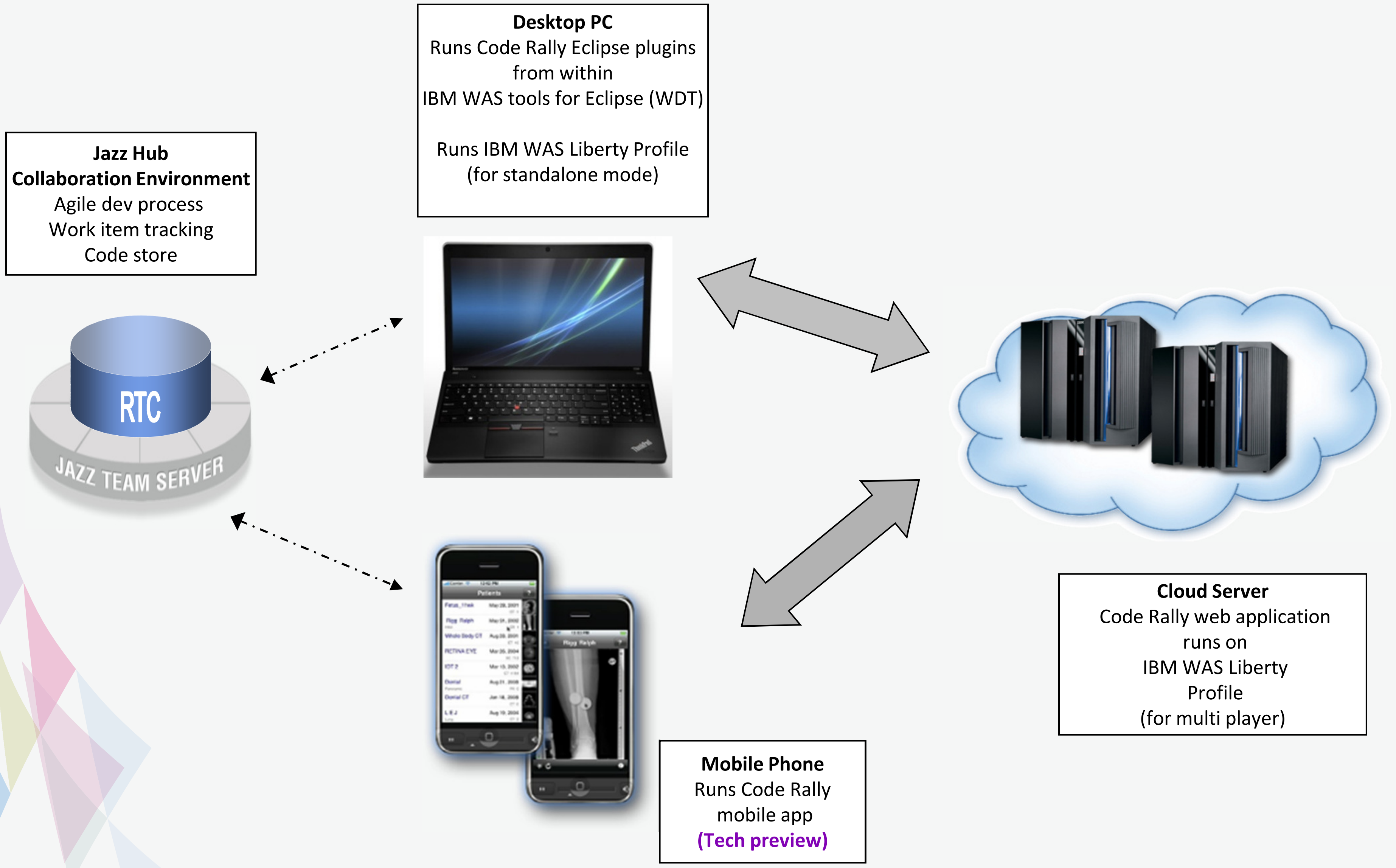
- Eclipse 4.2.2 (eclipse.org) with the Rational Team Concert plugin
- WebSphere Application Server Developer Tools – Liberty profile (Eclipse marketplace – marketplace.eclispe.org OR wasdev.net)
- Liberty runtime (wasdev.net)
- IBM Worklight Mobile Application Solution(http://www-01.ibm.com/software/mobile-solutions/worklight/)

Feedback is always welcome at our Code Rally Forum:

https://www.ibm.com/developerworks/mydeveloperworks/blogs/code-rally/?lang=en

Or the jazz hub for enhancements or defects: https://hub.jazz.net/project/Code%20Rally

Code Rally Development & Deployment Environment



Desktop PC
Runs Code Rally Eclipse plugins from within
IBM WAS tools for Eclipse (WDT)

Runs IBM WAS Liberty Profile
(for standalone mode)

Jazz Hub
Collaboration Environment
Agile dev process
Work item tracking
Code store

Cloud Server
Code Rally web application
runs on
IBM WAS Liberty
Profile
(for multi player)

Mobile Phone
Runs Code Rally
mobile app
(Tech preview)

Development of the Game

The globally diverse team consisted of: game developers (3 students), UI developers, a development team leader, product manager, marketing manager, and a project manager.

The team used 2 week iterations to develop the game. (half a day for planning, 7.5 days for dev, 2 for testing, a demo to the team at the end of the iteration).

A weekly project meeting was held to discuss the overall status of the game including development, testing, graphics, documentation, samples and legal details.

A daily scrum meeting was help between the game developers and the development team leader to discuss the progress of the features for that iteration.

Planning the iteration

At the beginning of the release cycle, the team put together a number of stories in jazz hub to describe the features of the game and then assigned work items for each story.

At the beginning of each iteration we would have a meeting where we would plan the iteration. The work items were reviewed and prioritized based on building the infrastructure of the game first , followed by nice to have capabilities. Defects and stories were treated equally and were prioritised based on the impact on the end user.

Each story had its own point value representing the number of ideal developer days we estimated the work would take – this point value was assigned in agreement between development lead and the developers. These point values helped us assign an appropriate amount of work to each iteration.

Due to the development team being split between the US and UK we held meetings in the UK afternoon to facilitate the difference in time zones. Any issues occurring outside the overlapping work hours would be dealt with via email.

Implementing the work items

The developer team worked on individual laptops (typical config: dual core CPU, 2GB memory, 150 GB hard drive). Each developer had their own copy of the development environment (Eclipse, WDT and Liberty).

Developers ran builds locally on demand as it took <1 minute to build the project on the development machines. Code was checked-in to the RTC server throughout the day as a backup and was delivered to the stream when a functional component of the work was complete.

As the game grows more complex and we start implementing automated testing we have migrated to an automated build process capable of running test suites for us – all within RTC.

Testing during the iteration

After 8 days of development, the team moved into testing and bug fixing mode.

Due to the complexity of the game, the development team manually tested the code rally game. We could have automated from the start but as we did not the technical debt built up.

This greatly impacted the ability to regression test new changes efficiently. Each change need to have a series of regression tests manually run and verified.

Intermittent defects were able to get past our testing and we had no basis for automating multiplatform testing

Demo of the work items implemented

At the end of the iteration, the development team did a demo to the broader team and stakeholders.

The demo highlighted the new capabilities that were implemented in the current iteration.

Feedback from the broader team was provided and any usability or capability suggestions were added as work items for consideration for future iterations.

This final iteration driver was made available for internal groups to use or evaluate.

Specific content: Tracks for the game

The first impression of the game is the track.

The track had to be:

- appealing
- easy to see and interpret the track
- not be to overwhelming

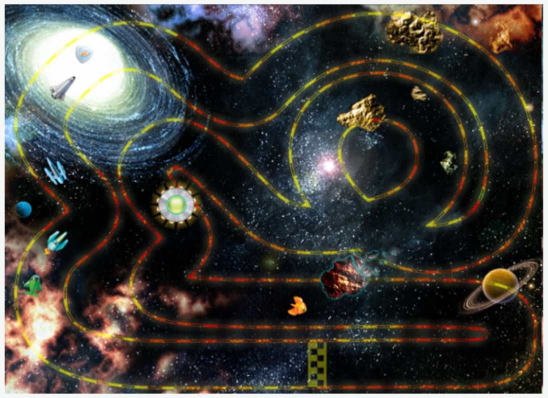
- The team decided to provide many different tracks so it would appeal to a wide audience.

- Some people prefer the typical oval race car track, other prefer a space track, a swamp, flying in the sky etc.

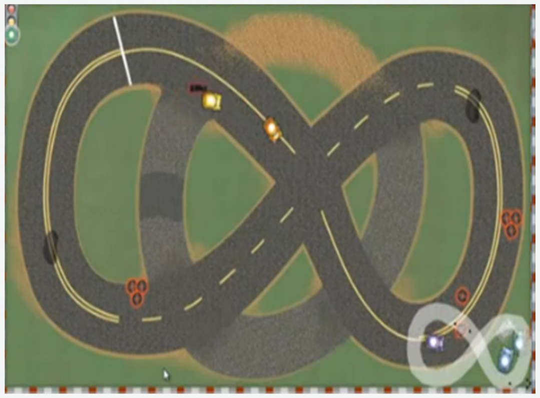
Which track do you want to play on?

Different tracks that are available for the user to select:

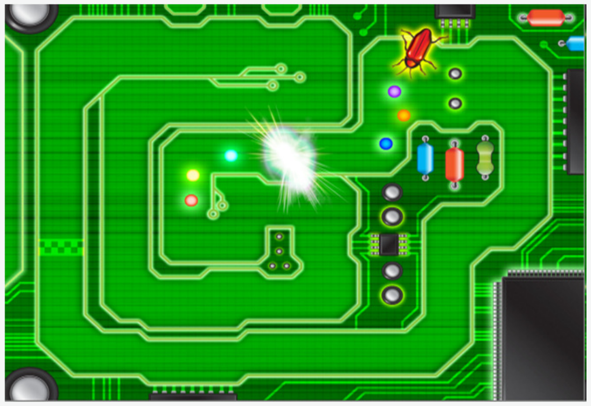
Space track



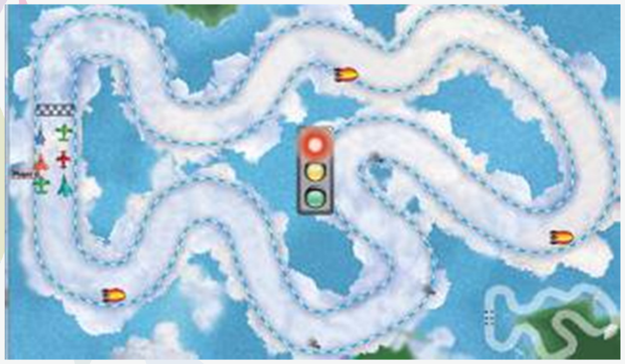
Race track



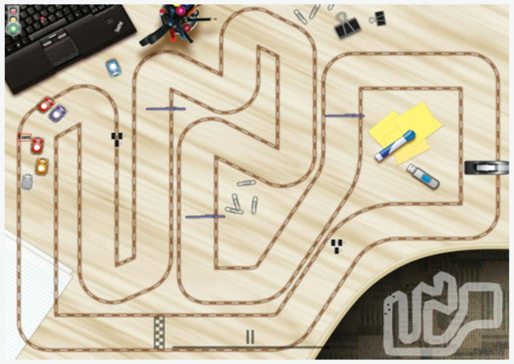
Circuit board track



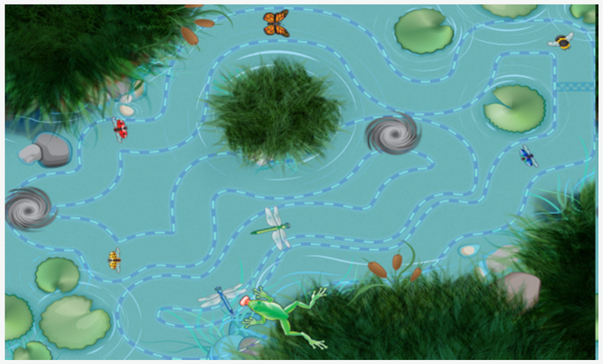
Cloud track



Desk track

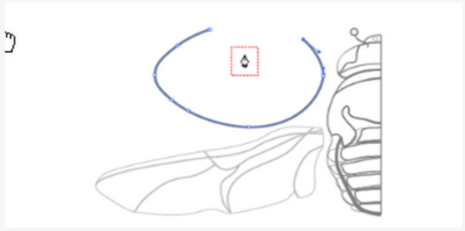


Pond track

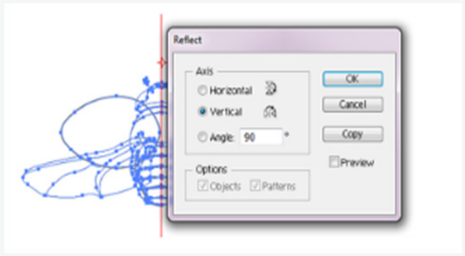


How the graphics were built

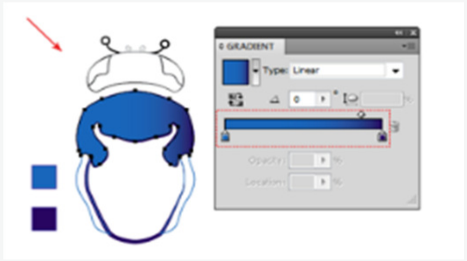
Step 1: Draw the graphic by hand and then trace digitally input into Adobe Illustrator.



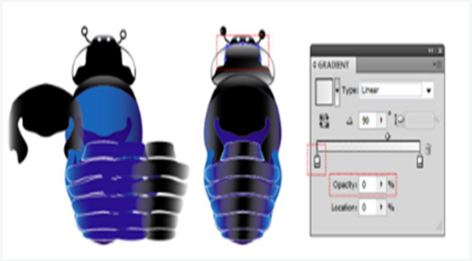
Step 2: Once half the object is created, use the "Reflect" action to make a mirrored copy.



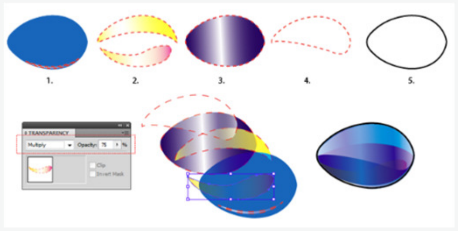
Step 3: Hide parts of the insect and focus on adding color and determining highlights and shadows.



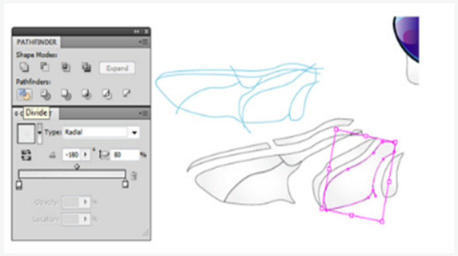
Step 4:Layering the color and gradients of the insect allows it to blend nicely when stacked.



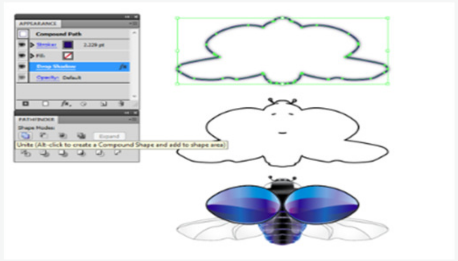
Step 5: Using "Transparency" , allows the blending mode and opacity to create a different effect.



Step 6: By extending the inner lines across the outermost line, the divide tool allows separate of the segments in the wings.

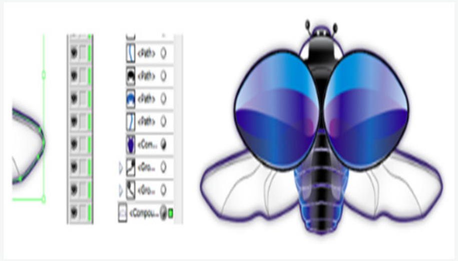


Step 7: Once all the components are colored, we can use "unite" to bring the insect together.



Step 8: Clean up some of the loose lines, thicken the stroke and merge the layers!

Voila – the blue ladybug!



Challenges: Racing the car

The team needed to determine how the car will be raced around the track. To that end, the car starts at an initial position and needs to be told where to aim and what to use for acceleration and breaking values.

The racetrack has a series of invisible “checkpoints” which span across the width of the road that the vehicle must move between in order (much like Olympic skiing). The user must aim for a point along the next checkpoint to get the car to drive through it.

The event driven car API detects and calls methods for:

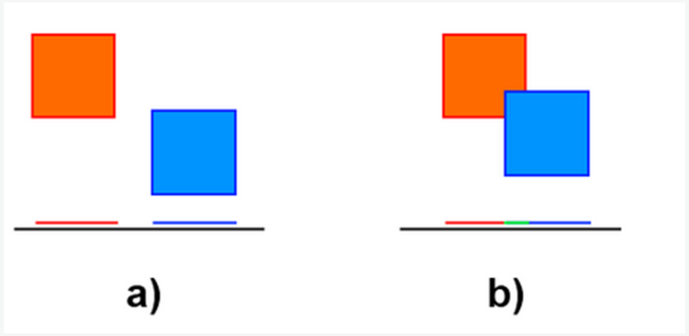
- Obstacle collision
- Other car collision
- Being off track
- Proximity to other cars
- Proximity to other obstacles
- Stall

Challenges – How to detect collisions

The team used the Separating Axis Theorem to do collision detection in the game.

If two convex objects are not intersecting, then there exists an axis for which the projections of the two objects do not overlap.

To picture this concept of "overlapping projections", look at the image below. In diagram **a** you see two squares which are clearly not intersecting. If you were to project their respective images onto the horizontal axis as pictured, their projections would not intersect either. As soon as you have found an axis in which these projections do not overlap, you can assert that the objects do not intersect. Conversely, the two squares are clearly intersecting in diagram **b**.



Challenges – How to detect collisions

Unlike the previous image, objects being tested for collision won't usually be simple, un-rotated squares. You might be dealing with a complex polygon with many vertices. You may be wondering then, what the best way to programmatically project these polygons onto an axis. You would simply loop over all of a polygon's vertices, perform a dot product between the vertex and the normalized axis to be projected on, and store the minimum and maximum result.

When a collision occurs, the corresponding method will be called in the car AI – how the car reacts to the collision is up to the user.

Development & Project Management Stats

- Code Rally ear file: 20 MB
- UI Plugin for Eclipse: 8MB (mostly made up of car/ track graphics)
- How long it took to develop: 6 months
- How long it took to test: 6 weeks
- Challenges with testing: lack of automation – all testing was manual which meant our regression detection rate was poor
- Neat things we did with the code: we take source from a user and compile it on a server – this is impressive as we could create an IDE in a webpage with the server backend doing the compiling – you don’t even need the device you are developing in to be able to compile the code at this point. There are security concerns to doing this which is a challenge.
- Licensing and legal work: 3 months

Help us make Code Rally better

Should we publish an internet leader board?

Should we award achievements for certain scores? Certain activities (ie. # of car crashes, # of times off track ,…..)

Should we put together more tracks?

…

What's happening next?

We are continuing work on improving the game so please provide us input on what you'd like to see.

The Code Rally team has received interest from professors who would like to use the game as a fun way of teaching beginner-advanced Java techniques to their students.

The Code Rally team is looking to have hack-a-thons to enhance the game – our first is Sunday 24th March in New York. Check out our website for details.

IBM will continue to highlight the game at various conferences and forums (ie. WUG, Devoxx, WebSphere Impact, …)

Where do I find out more?

Code Rally Community Site:

This site provides you information on:

- Downloading and Playing the game
- Blogs and Forums to chat with other games and developer

Code Rally - A Developer Gaming Community

Welcome to the Code Rally community, the place to discuss, share and learn about developing for using IBM developer tools. In this community you will find the latest Code Rally news, as well as information and education directly aimed at software developers. Use the forums to discuss recent developments, share your opinions and get help from the experts.

[illegible]

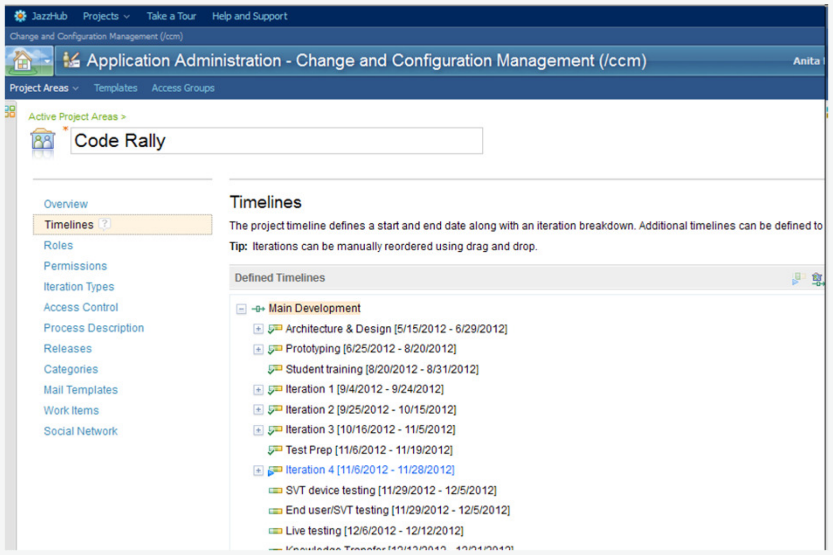
As we continue to build the game, we will share all milestones to help track our progress. Below is the latest milestone we'd like to share:

- [illegible]
- [illegible]
- [illegible]

Jazz Hub for Code Rally:

This site provides you information on:

- Agile development process used
- Development plan
- Features being planned



Code Rally

Timelines

The project timeline defines a start and end date along with an iteration breakdown. Additional timelines can be defined to

Tip: Iterations can be manually reordered using drag and drop.

Defined Timelines

- Main Development
  - Architecture & Design [5/15/2012 - 6/29/2012]
  - Prototyping [6/25/2012 - 8/20/2012]
  - Student training [8/20/2012 - 8/31/2012]
  - Iteration 1 [9/4/2012 - 9/24/2012]
  - Iteration 2 [9/25/2012 - 10/15/2012]
  - Iteration 3 [10/16/2012 - 11/5/2012]
  - Test Prep [11/6/2012 - 11/19/2012]
  - Iteration 4 [11/6/2012 - 11/28/2012]
  - SVT device testing [11/29/2012 - 12/5/2012]
  - End user/SVT testing [11/29/2012 - 12/5/2012]
  - Live testing [12/6/2012 - 12/12/2012]
  - Knowledge Transfer [12/12/2012 - 12/21/2012]

Backup

IBM tools and runtime used to build the game

- **Rational Team Concert** was used as the team development environment. Jazz Hub was used to open stories, defects and other work items and have them picked up and worked on by developers (and seen to be worked on) as well as having the code submitted with associations to the related work items made it easy to track how development was progressing.

- **WebSphere Application Server tools for Eclipse** was used to develop the web application and deploy to the WAS Liberty server.

- **WAS Liberty Profile** is the light weight application server used to allow quick, iterative development and deployment of the Code Rally application. The WAS Liberty profile server is also used to host Code Rally to handle all of the incoming and outgoing requests for running the game.