Building Tools for Empowering Creativity and Improving Efficiency

Charles Brandt
Senior Software Engineer, International Game Technology

GAME DEVELOPERS CONFERENCE™ CHINA
SHANGHAI INTERNATIONAL CONVENTION CENTER
SHANGHAI, CHINA · OCTOBER 19–21, 2014

- Engineers and artists are from very different backgrounds and communication isn’t always easy.

- By building tools for non-engineers to use tasks can be shifted from engineers.

- These tasks are better suited for non-engineers anyway.

Real World Case: DinerDash PHPExcel

- The goal is to externalize game economy variables in an excel document.

  - Source code: http://phpexcel.codeplex.com/

PHPExcel

- Diner Dash - restaurant management game

Benefits

- Game designers can enter variables themselves and save the engineers some time.
- Game variables are more human-readable.
- Designers can adjust variables quickly to fine tune the game and set up special temporary game events.

Difficulties

- Setting up the Excel files took a programmer.
- The format for the Excel sheets has to be strictly followed.
- Some files would not work and we eventually realized that one had been edited with Open Office and the file it saved was not compatible.

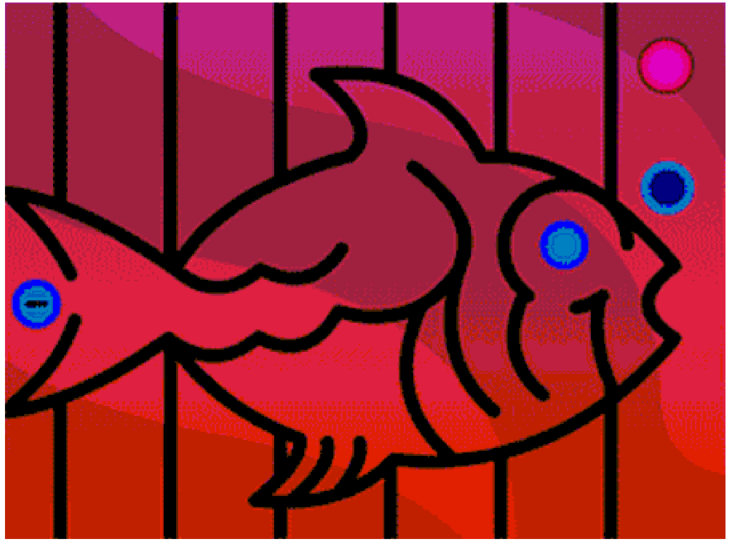
Was it worth it?

- At a point it seemed no but in the long term it was a very good idea. When a bug was found production stopped for about a day until we found the reason the file was not working.

- It is a very good idea to have a company-wide wiki to document these sorts of bugs and their solutions. Make posts easier to lookup by adding tags to them.

Real world case: Metamorphosis Image Analysis Collision Detection

- Analyze a small section of pixels at set intervals.
- When set colors are detected activate different types of collisions



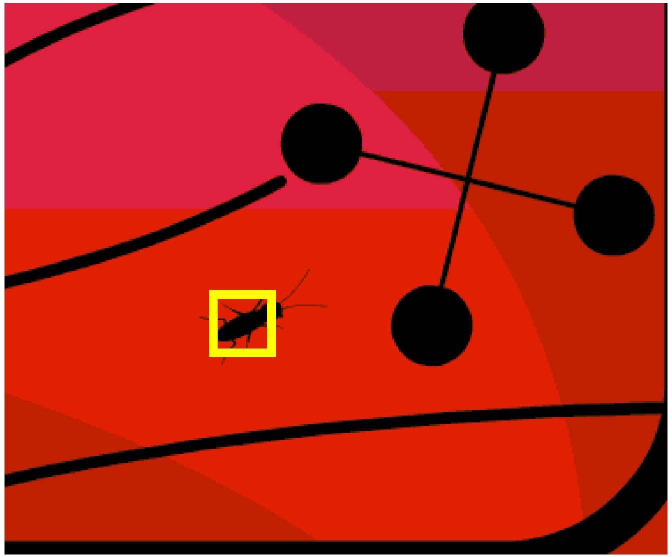
Benefits

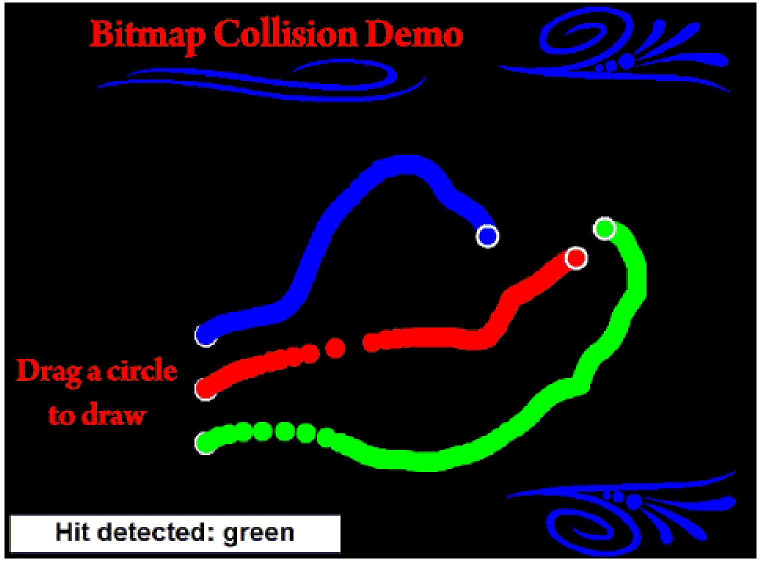
- Game designers can test level layouts very quickly without needing the programmer to update things for them.

- No need to program level by level. One collision detection engine can do everything.

Benefits continued

- New behaviors can be quickly added by just assigning new colors.





Difficulties

- Creating the initial engine can take some time to finesse to get running smoothly.

- Works best with few small detection areas.

Difficulties continued

- Movement made between detection cycles will be ignored. This is where some finessing is necessary to ensure accurate detection.

- Might not perform as well on slow machines.

Issues and solutions

- You do not want to use the same colors for detection as you wish to appear in the game.

- The solution for this is to make a detection map for your level. Save a copy of the level art and then color on where you want each detection behavior to occur using any color you want. This detection map is never actually shown on screen.

More issues and solutions

- Detection is running too slowly and you are not triggering collisions.

- Try increasing your detection rate and use some trial and error to find the sweet spot. You may try reducing the sample size of your detection area. Another way to speed up detection is to check every other pixel so you cover a large area using less calculations.

Real World Case: Covet Fashion Software generated YAML

- Game items, their prices, descriptions, etc. are all defined in a YAML document.
- Because YAML is strict with using spaces we do not want non-programmers to directly modify them.

Benefits

- Dramatically improves the speed and ease by which game developers are able to add and update game items.
- Designers never have to look at a line of code; instead they get a user-friendly GUI.
- Frees up programmers from an endless task of debugging constantly updated documents.

Difficulties

- Creating the software initially took two weeks for one programmer.
- During the course of creation it was decided that this tool should work with all games in the studio. Each game used different styles of formatting their YAML documents. This called for the need of a file to define schema for each game. On the plus side it led to setting a standardized format for future games.

Was it worth it?

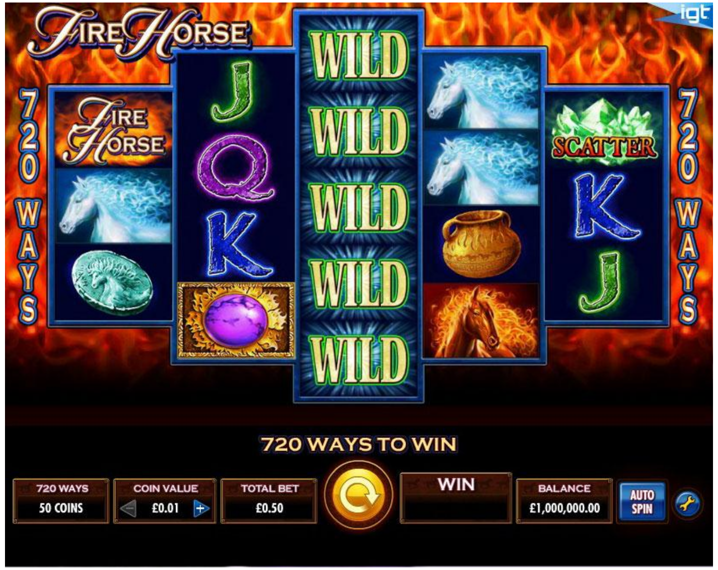
- Though it took several iterations of creating the software, it was well received by all the different studios.

- There was a inevitable need for the software and once it was created there was a lot less strain on developers from all the studios.

Real World Case: Fire Horse Gaff Tool

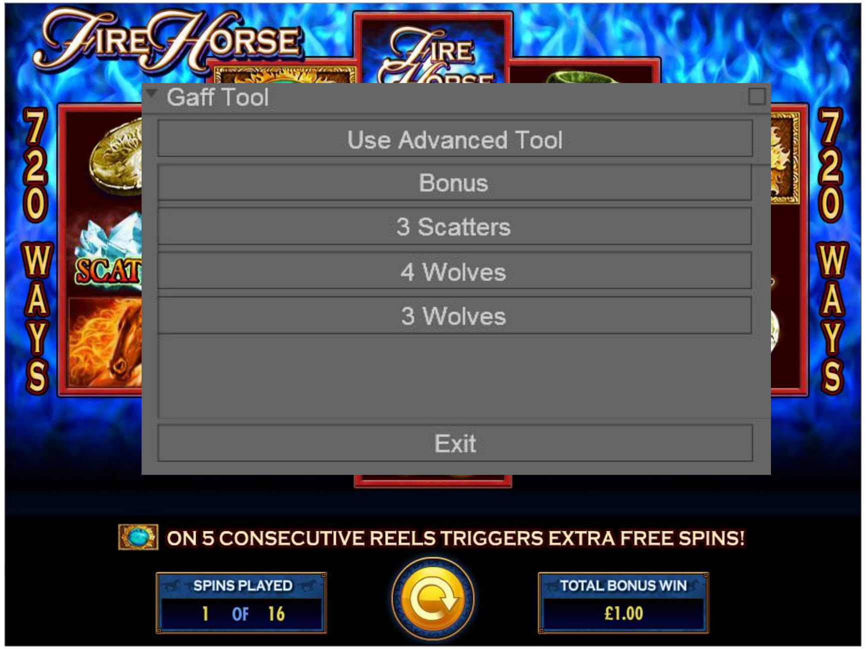
- When slot machine games are shown at trade shows and to customers we needed a way to show all the different features without having to wait for them to hit naturally.

- We added a new feature triggered by a button added to the existing interface.



Benefits

- The new functionality was a big success at trade shows.
- Adding new gaffs is done very easily by configuring an XML file.
- Once the feature was activated it proved to also be useful for testing bugs.



Difficulties

- A password was used for accessing the old way to force outcomes. Remembering this password has been a little tricky for some people.

- Once a 1.0 version was created there was great interest and more features were requested so a 2.0 version was released

Difficulties continued

- There have been a few questions about configuring the XML but this was solved with thorough documentation posted on the office wiki.

Was it worth it?

- Absolutely, previously potential customers would hire people to play the game until certain outcomes occurred. This is no longer necessary.
- The simple UI has proved to be easy to use while keeping the order of the reel symbols hidden.

Real World Example: Particle Editor

- Most artists do not know how to modify particle effect code and programmers are not the best people to create visual effects.
- It was decided software should be written to give artists control over particle effect creation.

Benefits

- Particles that have been used before can be reused and also used as a base for modified versions.
- Artists can create effects to their liking without going back-and-forth with a programmer.
- Particle effects are much more efficient for most games rather than hand animating effects

Difficulties

- An editor already exists but it is not intuitive and artists do not like it. What needs to be changed?

- How can we give artists a lot of power and still have them understand what everything does?

Was it worth it?

- This project is currently being developed but the alpha version was very well received by artists.
- We plan to use tooltips to explain the functionality of all the different options so it will be hidden until needed.
- The final version will export an effect file which the programmers will implement in the game.

Some guidelines to follow

- Build things generically. You never know what will be changed or added in the future.
- Build with a thought-out plan for how future modifications will be made.
- Version control your tools so you can always go back to an older version if necessary.

Guidelines continued

- Keep your tools self-contained so they can easily be integrated in to something else.
- Write your own documentation and put it in an easily accessible location. Your job is not over until you do this step.
- Keep projects smaller in scope if possible.
- Limit the number of programmers. Less programmers will ensure less confusion in initial development.

Problems that will arise

- Once people start to see the great tool you are making they all start to ask for custom behavior. There is a lot of change at this stage so make sure your code is well organized so your code doesn’t become unintelligible by the end of the project.

Thank you for listening!

Are there any questions?

- Charles Brandt
  - Email: brandtcharlesc@gmail.com
  - Website: sites.google.com/site/charlesbrandtportfolio/