GAMASUTRA
The Art & Business of Making Games

PRINT

Indie Postmortem: *Armadillo Run*

By Peter Stock

Armadillo Run is a puzzle game, with gameplay based on realistic physics simulation. It's like virtual Meccano/Lego, with the goal of getting a ball to a target. It's a low-budget, independently developed game, which I have self-published on-line. I have a background in programming and a degree in computer science, but no previous game development experience.

The Origins Of Armadillo Run

I decided to try my hand at making a game in July 2005. I thought that since I had a very low budget, the best strategy would be to make something small, interesting and different. I briefly thought about the target market (downloadable PC games) and thought that a slower-paced game with a mouse interface was more suitable than a joypad-based action game (which would be more suitable for a console). I wanted to work on something that interested me, and thought physics would fit the bill.

Introduction

I started off making a 2D spring simulation. Springs are great because they are simple to simulate individually, but what's interesting is that you can put them together and get a lot of interesting behaviour 'for free'. After spending a couple of weeks exploring the simulation of springs, I decided on the idea for my game - constructing structures from springs to transport a ball across each level of 'obstacles'. There was just one more thing I needed to make it a game - an animal that bore no resemblance to its physical counterpart (I've never seen a blue hedgehog). An armadillo seemed appropriate because one species of armadillo can roll itself into a ball. After my best artistic efforts, I had my mascot.

I had finished the bulk of the game in 4 months (keeping to my development schedule pretty well), but it then took another 5 months until the release (time which I hadn't planned for). I'm glad I took the time to improve it, but I vastly underestimated how long it would take.

What Went Right?

1. It Was A Good Idea. The idea behind the game was fun and sufficiently different from other games to make it unique. In the competitive games market, I knew it would only be worth making if it was fundamentally different - there's no way someone like me can compete with the visuals of modern games. The physics simulation and novel gameplay are the unique selling points, making the relatively unsophisticated graphics less important.

It was definitely a case of the game concept being created around the technology though, rather than technology being created to fit the game concept. *"I think I can do some physics stuff - how can I make this into a game?"* was my initial train of thought. This approach to game design has been criticised, but it does remove some of the development risks.

2. It Was A Small-Scale Low-Budget Project I planned from the start to keep things simple and cheap. This was my first game and judging from other people's experiences, I knew I should keep costs low and not bite off more than I can chew. If the project totally failed, pretty much the only loss would have been my time. While this attitude might be counterproductive for established studios, I think it's pretty sensible for small-time chaps like me.

I took the step of leaving full-time employment while I developed the game. Development is more rapid when you can devote 8 hours a day to it rather than 1 or 2 in the evenings and weekends - you aren't as tired and don't have to remind yourself where you left off. I think this 'burn the boats' approach can work well, but setting and achieving targets becomes more of a serious issue when starting out on this road because your development time is limited by your bank account!

3. The Implementation Was Done In The Right Order. The core of the game was the physics simulation. If there were problems implementing it or if it was too slow, then the game idea would have had to be changed or abandoned. I'm sure everyone sometimes thinks

"but I need to properly develop the graphics engine before I can write that" or *"yes, but wouldn't it be cool to do ... first"*but I tried to ignore these particular voices and push on with what I knew I needed to get sorted first.

Thankfully, the physics turned out to be quite manageable so development carried on to include the next most important/critical things for the gameplay. I had most of the core code written in a few weeks, which was very motivating.

4. Good Work Environment. The games industry is notorious for its poor work practices and long hours. I've never worked for a games company, but you don't even need to read between the lines to know that this reputation is not wholly unjustified. I resolved to make sure my working environment was reasonable.

I found working from home requires some discipline, both getting down to work when there are distractions as well as leaving your work in the evening when there's something you've want to get finished. I tried to make sure that I worked at least 35 hours a week, but not more than 50 (everyone's less efficient when they are tired). I also set aside some time in my schedule for days off. On the whole I kept to this, and I feel it worked well for me - I don't think I would have finished quicker if I had worked longer hours.

5. Testing And Feedback. I read a few books and articles on independent game development before and during this project. They advocate testing - early and often. I am a bit of a 'plodder' when it comes to writing code, but I don't think this is a bad thing (fewer bugs = less time bug fixing). I also make sure that I test new things as I write them - nothing formal, but I do at least try to break them (isn't that the point of testing?).

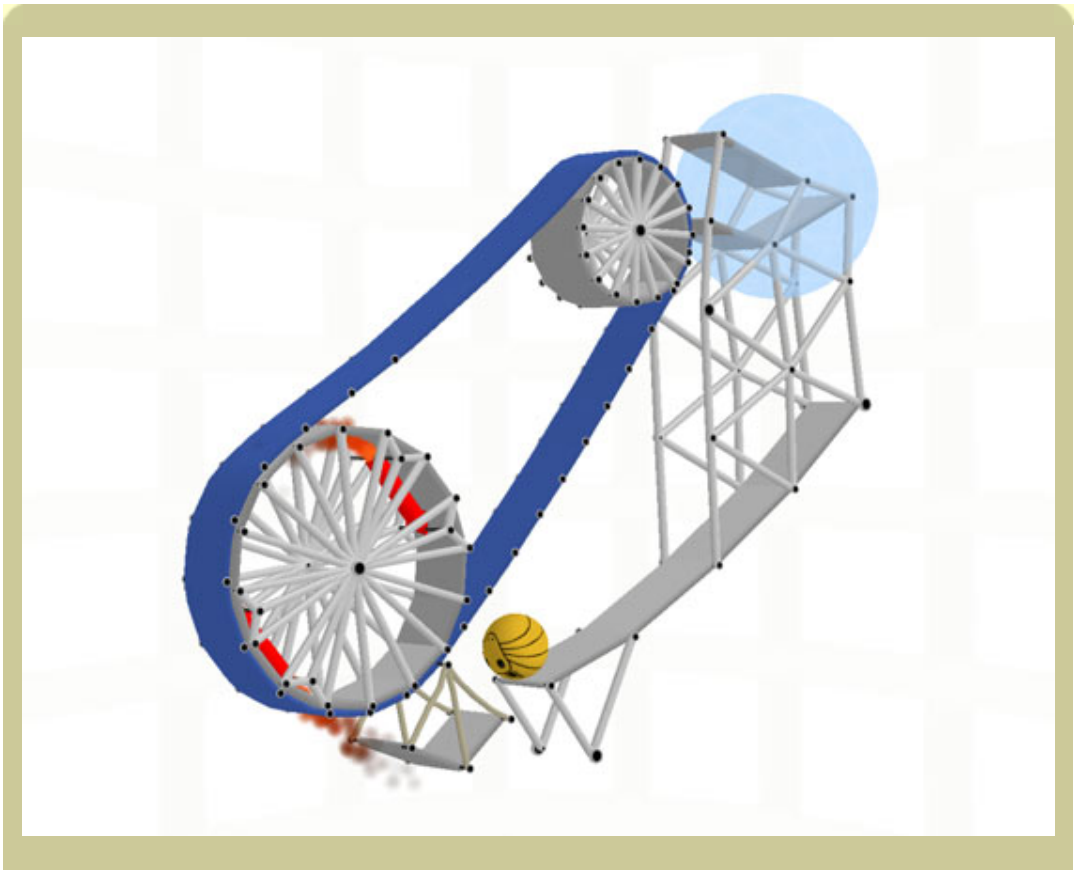
Anyway, I read that playtesting is even more important - as well as acting on the feedback you get. Actually the book "The Indie Game Development Survival Guide" stresses the point that negative feedback is better than positive feedback, since you can use it as a tool to improve your game.

Many important usability problems have been raised by people testing my game, which I didn't notice myself - I was surprised how often I had made bad choices. Here are a couple of examples of where testing and feedback radically improved the game:

- **Awkward editing** - Flexible links are just simulated as chains of small links, and were originally edited as such. After playing an early test build, one of my friends pointed out that this made the editing painful. It wasn't until then that I realised the system was totally awful and changed it to make a simpler editing interface (but a more complicated implementation).
- **Tooltips** - Another one of my friends pointed out that the interface buttons would really benefit from having tooltips on them. Having created the buttons and knowing exactly what they did, this hadn't occurred to me.

So I always make sure I listen to comments now!

"You're all wrong and all the kids are right."
-Local H



Armadillo Run

What Went Wrong? 1. Poor Time Estimation. One of the first things I did was draw up a work schedule covering what I thought was the whole development. This was good - it helped organise the work and gave me short and long term targets. I was quite optimistic in my time estimates of most tasks, and for the most part achieved my self-imposed targets.

However, I didn't allow enough time for the things I viewed as less important - which turned out to take significantly longer than I expected. I think this was because I'm a programmer, and I thought that programming a working game engine was pretty much all you needed to do to make a game. Sound, interface design, level design, testing, gameplay tuning, menu code and the general finishing touches all got massively underestimated. Thankfully my 'slow but steady' coding style didn't create a mountain of bugs on top of these delays.

The other thing that I completely ignored was publishing the game. I did give it some thought early on, but I didn't allocate any time for it. I had assumed I would just give it to someone else to take care of that bit. In the end I decided to sell it myself through my own web site, but setting it up took some time I hadn't planned for.

2. No Publishing Strategy At The Beginning. It wasn't until I had begun playtesting that I gave significant thought to publishing the game. I had three main options:

1. Sell the game on my own web site.
2. Work with a web-based publisher on a royalty-based deal.
3. Sell the distribution rights for my game for a fixed sum.

Obviously the second and third options are predicated on finding a willing publishing partner. The third option would likely be bad for me because I wouldn't have much leverage when making a deal due to my inexperience.

I was initially put off from selling via my own web site for two reasons - handling payments and marketing. Fortunately there are numerous third-party payment processing options that charge a low fee. After a bit of thought, I figured the best (and cheapest) kind of marketing is word-of-mouth from satisfied customers and I asked myself *"what would a publisher do regarding marketing that I couldn't?"*

So I decided on the first option - selling it on my own web site. A decision which probably should have been made earlier so I could have planned it better and built up some interest with a pre-release demo.

3. Premature Optimisation. I don't like to write throwaway code - I think it's better to take the time to do things properly the first time. But sometimes I took this too far and spent a little bit too long optimising parts of the code that were still in a state of flux.

Of course, getting the simulation running efficiently was on the critical path (without this I would have had to reassess the project), so I needed to do some of the optimisations early. But once it became clear that the project was feasible, I still went back and twiddled with things to make them better. Maybe this was for the best because the ideas were still fresh in my mind, but it did push my schedule back a little.

I should mention that some of my ideas for 'optimisations' did in fact make the code run slower, which highlighted the importance of profiling - both to locate candidate code for optimisation and to test the optimisations. The book "Game Programming Gems" has a good article on an in-game profiler that I found helpful.

4. Artwork. I'm no Michelangelo, so I decided early on to keep artistic requirements to a minimum since I would be the one doing any artwork. I decided on a 'clean' style that didn't use many textures, partly to keep the screen uncluttered, but partly because I don't have the artistic talent to do anything better.

Looking back, I think maybe I should have spent some money and hired an artist to do some work on the game, but I hadn't budgeted for this and I'd overrun on my schedule, so I decided against it.

5. No Design Document. Seeing as it's a small, technology-based game, I didn't think it was necessary to have a design document. In fact it started off just as an experiment with physics. Only when I had successfully implemented the simulation code did I build a more concrete

idea of how the game would play. In retrospect, I should have taken some time away from development at this point to structure my ideas on paper and do interface design and preliminary level design. The implementation of some parts of the game was a little directionless, evolving rather than being intelligently designed.

Conclusion

Overall, I think the development of Armadillo Run went well - surprisingly so given that it's my first game. I partly attribute this to the research I did on independent games development before I started out, where I learnt several lessons without having to make the mistakes myself. I hope this article helps other people in the same way.

Game Data

Publisher: Peter Stock
Developer: Peter Stock
Number of full time developers: 1
Length of Development: Nine months
Release Date: April 22, 2006

References

Game Programming Gems, Charles River Media, 2000
The Indie Game Development Survival Guide, Charles River Media, 2003

Return to the full version of this article