Postmortem: Avalanche Studios' *Renegade Ops*

By Axel Lindberg

It all started for Avalanche Studios during the later phase of the*Just Cause 2* beta. The bug reports were still pouring in from the QA departments. This was no time for cool ideas and awesome features. In fact, the very purpose of "beta" as a development phase is to have a team of enthusiastic and creative wizards mutate into a disciplined regiment of bug-fixing soldiers.

The job is no longer about feeding the dream of what the game could be -- it is about coming to terms with what the game is not. Beta is a necessary battle that every game developer must go through - for without beta, a game will behave strangely, crash, and be generally unplayable.

In the midst of this battle, I was working on a particularly annoying bug -- when suddenly, I was called into a meeting... "Hey guys! How would you feel about getting started on a new project? Sega is interested in seeing a concept pitch for an XBLA/PSN/STEAM game with an original IP, using our engine. What do you say?"

Our answer was somewhere along the lines of "OMG! Rock 'n roll!" or similar. We were thrilled.

Why *Renegade Ops*?

The first thing we did before getting to work on a concept for the game was to sit down and have a look at what games were currently out on Xbox Live Arcade and the PlayStation Network.

This was around the time when Chair Entertainment released*Shadow Complex*. We fell in love with it. The developers had done a fantastic job of delivering a modern take on Nintendo's classic *Metroid* games for the NES and SNES consoles, redone with Epic's Unreal Engine.

The IP was new, and a lot of the mechanics had been updated and deepened to fit the expectations of today's gamer audience -- but the core philosophies and concepts that made *Metroid* so awesome back in the '80s were kept perfectly intact. The sense of nostalgia it created in us had us sold.

We wanted to base our game on the same idea -- bringing a concept from the "golden era of gaming" into 2011 with a modern spin on it.

We had many discussions about what games we would take our inspiration from. The project would be very small in scale compared to big AAA-budget games, so we needed to choose a source of inspiration that played to Avalanche Studios' strengths and experience: over-the-top action, big destruction, vehicular mayhem, and a tongue-in-cheek-mega-macho attitude.

After we had this down, it didn't take long to agree upon our main sources of inspiration:*Jackal* (NES 1988), *Desert Strike* (Genesis/Mega Drive 1992), and the old Saturday morning GI Joe cartoons fit the bill perfectly. This provided a clear framework to work within and helped unify the team's vision for the game from a very early stage.

Postmortem

The primary goals that had been set prior to the projects start were fairly straightforward, though they would certainly prove to be a challenge to deliver on.

- The game should look, sound, and feel like an AAA-budget game, only in a small package.
- It should give us a foundation for online tech within the Avalanche Engine (multiplayer).
- It should reach an 85 percent metacritic score.

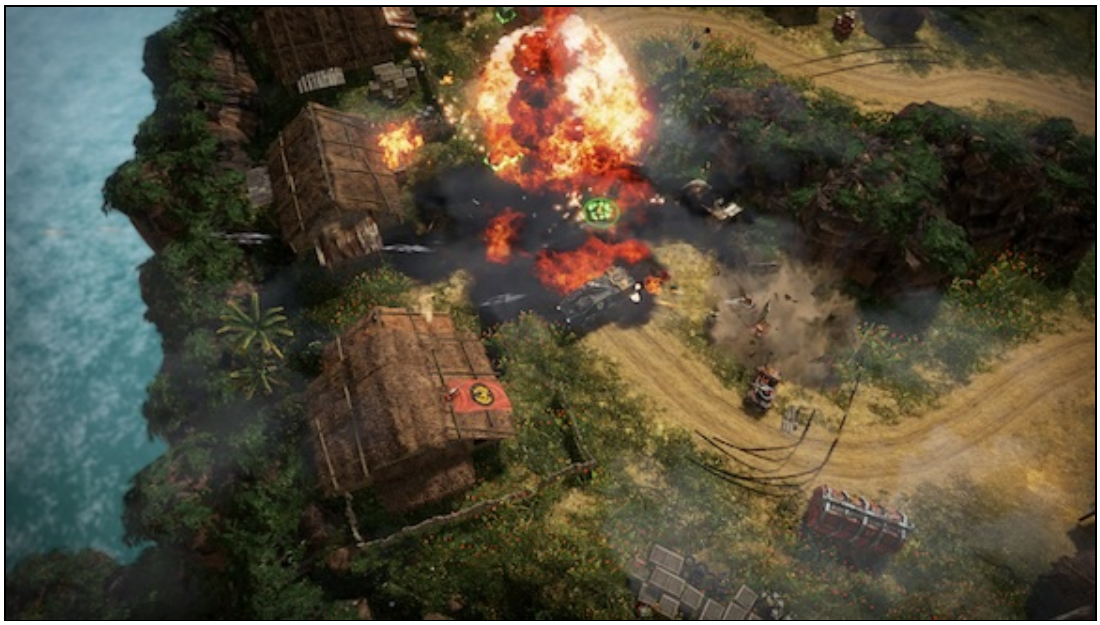
Reading through the reviews for *Renegade Ops*, it seems we succeeded with the first goal. Tom Chick writes the following in 1UP's review of the game: *"...it [Renegade Ops] manages to capture all the over-the-top action-movie glee of Just Cause 2. But it does it within the confines of a top-down twin-stick shooter..."*

The second goal was also accomplished. Multiplayer exists in the Avalanche Engine now, and I can't wait to share where that door is leading

us in future projects. Unfortunately, if I shared that information with you today, I wouldn't be able to share *anything* in the future.

In regards to the third goal, *Renegade Ops* ended up with a Metacritic of 81. I am happy with the score, and feel it is a fair number to land on when I attempt to look at the game objectively. However, since the goal was set at 85, this is clearly something we failed to deliver on.

This is all well and good on the surface, but let's dig a little deeper into the juicy bits and bobs of what it was like developing *Renegade Ops*. What was awesome? What sucked? What manner of life did we lead in the midst of milestone deliveries and beta submissions? Read on!



What Went Right

Every project is an opportunity for evolution of both the process and the individuals involved. This post mortem is an attempt to capitalize on that opportunity and share it with others. I hope this section will provide something interesting or useful to that end. Let us begin with the major aspects of our approach that we feel proved successful.

1. Build-Centric Development

The primary goal for *Renegade Ops* was that it had to be a fun game to play. It became our highest priority and we discussed "fun" a lot in the team. We wanted to find an approach that naturally let "fun" be the main factor of all decisions. The most reliable tool a developer has to accomplish this is to play the game *lots* during development, and to openly discuss any issues or gaps that hamper the "fun".

The term we used internally for our approach was "Build-centric development with focus on core mechanics". First of all, this means that we must always have a working build of the game available. As soon as there were any crashes or similar issues that made it impossible to play the game, the team dropped everything and fixed it.

We also let the latest version of the game be the deciding factor on what we were going to work on next. Instead of relying primarily on documentation for that purpose, as is the classic practice. We played the game and basically said things like: "Okay! The driving and shooting feels great now! I'm sick of shooting jeeps, though. We need to get a bigger more interesting enemy in there and see if we can make the combat more exciting and varied!"

Then we looked at our high-level design document, which had a list of potential enemy types and said: "A tank! Big and scary! Perfect! Let's get started on it next week!"

Now, don't get me wrong -- we did have an overarching plan for the project with defined milestones and commitments toward the publisher, but we made sure they were formulated in a way that gave us as much room as Sega was comfortable with, to allow us to follow our instincts (thanks go to Sega!) The milestone definition could, for example, read "Implement two enemy types", but it didn't specify which enemies out of the ones listed in our high-level design doc we would end up delivering.

---

2. Minimize Documentation

So what is all this fuss about? Well, developing games is a very complex and somewhat chaotic process, as is the case when working with any product that is driven by vague terms like "fun", "intuitive", and "immersive". The problem with relying on documentation too heavily is that game development is a super-iterative process by nature. No matter how good the design is on paper, or in theory -- the process of getting those ideas to work as envisioned in the actual game is one of refinement through trial and error. In addition to this, any single, seemingly trivial change to a gameplay system potentially (and more often than not) has a huge effect on the game as a whole.

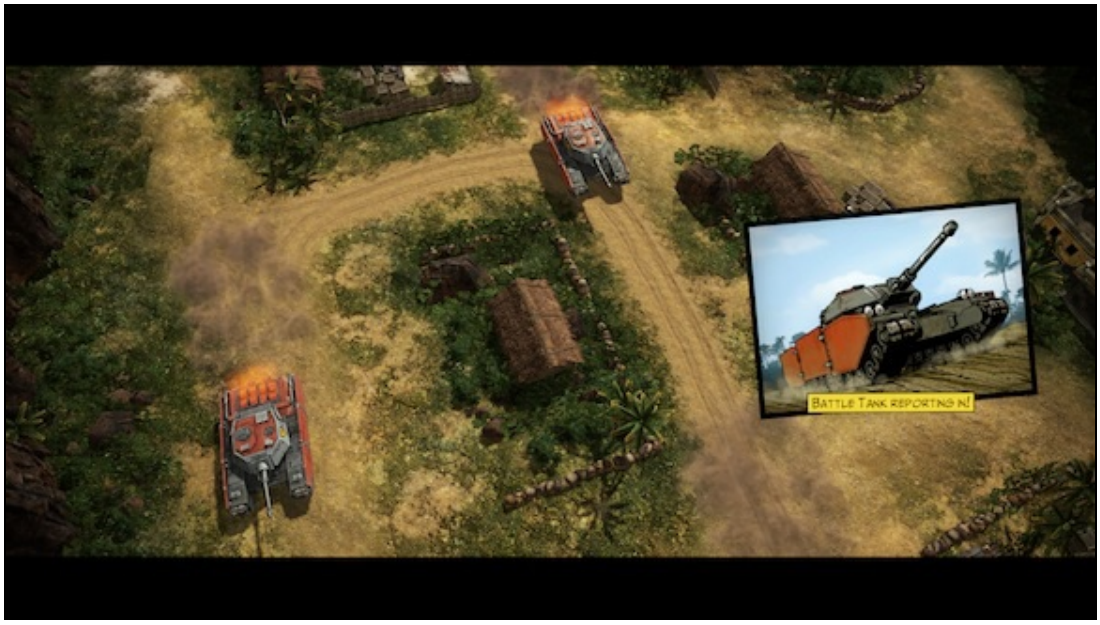
These factors largely counteract the notion of pouring time into the creation of awesome documents, since those documents usually become outdated the instant those ideas are tested in the game. This can lead to people spending a majority of their time rewriting and updating documents, instead of making the actual game!

We put as little time and energy as possible into documentation -- instead relying on verbal communication as our primary tool of communication and knowledge spreading, both internally and externally. As long as the documentation provided a high-level idea of the game and how it would play, we dropped it like a hot potato and got back to work.

Internally, we worked to create a culture within the team where open, free, and spontaneous verbal communication became a natural part of our collaborative. If anything ever was unclear to anyone, we preferred they took the time to talk about it face-to-face, rather than checking a document on the server that may -- or may not -- be up-to-date (and if they are up-to-date, that means someone is spending a lot of time updating all the documentation on a daily basis -- so either way it's a loss, in my eyes).

Externally, we were fortunate enough to have a very healthy, open relationship with Sega. We had regular conference calls with the publisher several times a week where the floor was open for any discussions to take place. Whenever a more pressing matter needed attention, we preferred to speak about it over the phone, rather than mailing each other about it.

In addition to this, any milestones that included any major documentation were accompanied by an audiovisual recording where I would talk about the process the team had gone through to arrive at the ideas and decisions presented in that documentation. This allowed us to maintain a very manageable level of detail on the actual documents themselves, as I went in to the gritty details verbally instead -- and it was material they had constant access to and could share with other stakeholders.



3. Focus on Core Mechanics

Another important decision we made from the very beginning of the project was to put a lot of effort into the core mechanics of the game. We simply didn't have the manpower to produce an endless amount of features to an acceptable level of quality, so we needed to have a neat little package of core mechanics that would hold up for as long as possible instead. Regardless of the scope of the project, I believe that the core mechanics of any game are the most important factors in making a great game.

We defined core mechanics as the simplest possible definition of the game in terms of mechanics. With*Renegade Ops*, we had decided to do a top-down vehicular shooter, so the core mechanics were fairly straightforward to discern right there: a top-down camera perspective, driving vehicles, and fighting enemies. The bare necessities of each of those three mechanics would make up our core mechanics. Every other mechanic and feature in the game would then support that core in one or several ways -- such as upgrades, narrative, missions, etc.

This is how we defined the core mechanics of*Renegade Ops*:

Core Mechanics

1. Player Perspective (Top-down view, Camera placement and functionality)
2. Primary Interaction (Vehicle driving controls, Aiming / Shooting)
3. Primary Challenge (Enemies, Health system, Destruction, Health pick-ups)

The narrative, however, is not a core mechanic (the story in Renegade Ops is presented using 2D comic art panels that fly in off-screen while the characters dialogues can be heard) and it is a good example of where we chose to put a lot of focus and energy on something that really was not a part of the core mechanics, and it cost us quite a bit of production.

I like how the narrative turned out -- I especially love the 2D art that Giorgio Cantú produced for the game -- and feel it injects a healthy dose of flavor and attitude into the game as a whole. But if we were to turn back time, we would have most likely gone for a simpler solution in regards to the game's narrative.

That being said, we still managed to have a strong focus and succeeded in delivering a set of core mechanics that we are happy with. Succeeding with core mechanics is very much a matter of priority. We agreed that we would not put any work on upgrade systems, secondary weapons, button-mash sequences, or similar -- until the basic driving, shooting, and enemy behaviors were at a level that was comparable to the competition.

---

4. Focus on Measurable Success

In addition to focusing on the core mechanics of the game, we also put a lot of energy into making sure every person on the team was working on tasks that had concrete, measurable results.

When making plans for each milestone, we asked ourselves how every chunk of work on that list would improve the quality of the game in a defined and measurable way -- not just for the publisher (which is standard practice) but for the *team*.

We strived to prioritize our production in a way that would get the team more excited about -- and confident in -- the project. This approach gave everyone (the team, company, and publisher) a strong sense of confidence in the project early on.

This had the unexpected effect of feeding a steady stream of positivity into the team, which lead to very high morale throughout its development. Long before our vertical slice was submitted, we were saying things like: "Damn! The game is already so much fun to play! This is gonna rock! I can't wait to get the stuff I'm working on now, into the game!"

Our vertical slice was a huge success. It was the first time we had worked on a project that actually managed to deliver a true vertical slice -- a slice of the game that truly is representative of the final game. We chose the game's first mission for this, and in fact, we didn't change much about the first mission after vertical slice at all.

This upward spiral of positivity is something we are going to strive to make happen in future projects, as it is nothing short of awesome to be a part of when it happens. There is a catch, however, which I will get into when we get to What Went Wrong.

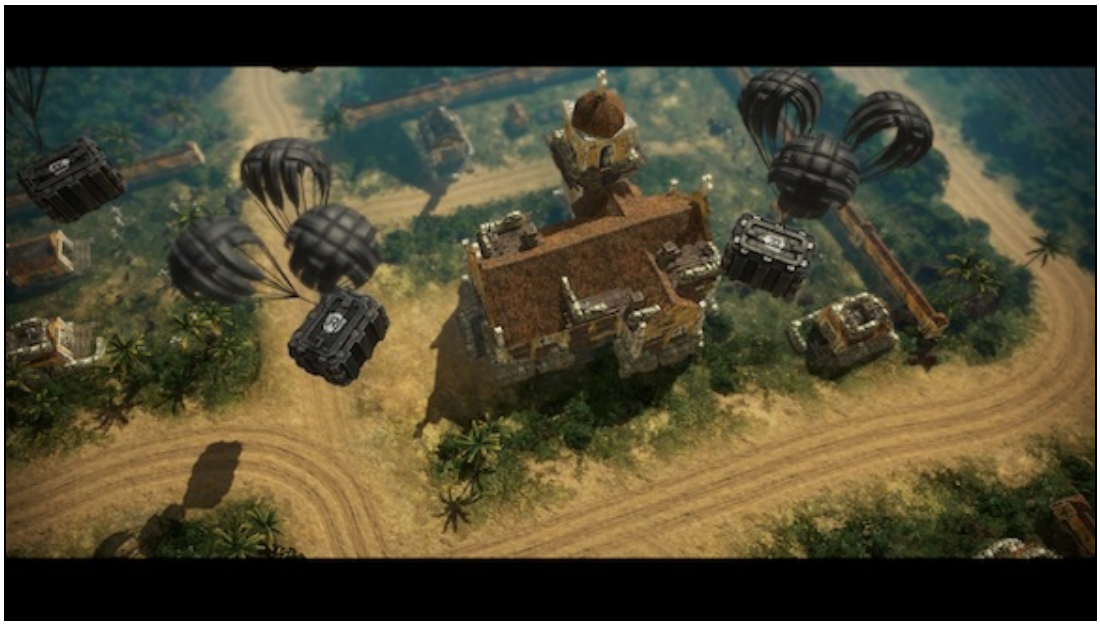
5. An Experienced Team

The *Renegade Ops* development team was built around a group of senior developers. The experience each individual brought to the table was a key factor in getting the project up on its feet and running in the right direction quickly, and consistently. We were eight people from the start, up until the main production phase, when we ramped up to 16, our maximum. The eight individuals who were involved from the start of the project had to function within several disciplines to cover all the facets of creating a game. Everyone pitched in wherever it was most needed, to the best of their abilities.

However, the most important factor of having a team of senior developers was that we could use our collective experience to mitigate risks and minimize waste by simply talking about a feature or system. Some of these discussions were time-consuming, but not nearly as time-consuming as it would have been to actually implement the system first and then go back to the drawing board once we had realized it wouldn't work.

These discussions also created an awareness of the limitations of each solution before it was applied -- an awareness that gave us time to work on adapting the game's design to function within those limitations right away, instead of waiting until after each system was in place.

Now hold on a minute! Didn't I just write about the "super-iterative" nature of game development and that our time should be spent on making the game? Yes, I did -- but that was in regards to documentation, not communication in general. Communication is paramount -- especially in regards to capitalizing on the experience that each individual brings to the team.



What Went Wrong

Beware, traveler, for the road grows dark ahead... Here are the major sins we committed and how they affected the team, the project, and the final game.

1. Greed: Gotta Have it All

Our ambitions for *Renegade Ops* were too high. We set out to make a game that felt like an triple-A budget game with a tiny team and a tiny budget in comparison, so the odds were against us from the start. However, compared to other XBLA/PSN titles, the budget was quite large.

In spite of this, we poured in an unacceptable amount of overtime on our own accord to pull it all together -- something Avalanche Studios' upper management frowned upon, since one of the company's primary goals is to avoid overtime at all costs. If we had been willing to compromise on the triple-A quality values to some degree, we would have had a bigger margin for error throughout the entire project.

One big problem with using the term "triple-A" to describe anything is that its definition is highly subjective. For some "triple-A" is only a matter of budget -- if you put more than X dollars into a game, it becomes a "triple-A" game. For others, it is critical reception; yet other see it as sales, or it is a mix of all of those.

Basically, every time anyone says "let's make a triple-A game," every person in the room most likely has their own idea of what that means. This became a problem for *Renegade Ops*, as Sega, Avalanche management, and the development team all had different views on what triple-A stood for. This created divergent expectations and priorities between us, which made it difficult to compromise, since everyone had different opinions on what was most important.

---

2. Lust: When Passion Goes Too Far

In regards to overtime, the team got so passionate and devoted to the game's success that it quite frankly went overboard. It was unhealthy. The lesson I have learned here is that, while a steady stream of positivity and upward spirals can be an awesome wave to ride, it can easily lead your team to a destructive place if left unchecked -- as expectations are constantly raised on all ends.

The more we strove to outdo ourselves with each milestone we delivered, the more pressure we put on ourselves to be even better. If the project had gone on for much longer than it did -- 18 months -- we would have crashed.

Luckily, that did not happen. Instead, the experience has given me a profound appreciation for developers that are able to ride that wave of positivity -- and create top-rated games -- all within normal working hours!

3. Gluttony: More Than Enough is Often Too Much

The scope for *Renegade Ops* from a feature standpoint was too big. At a glance, this seems similar to the first sin: that our high ambitions also caused us to go overboard with the amount of features we pushed into the final game. However, the symptoms tell a completely different story.

Our "lust" (passion and high ambitions) resulted in a core game experience with high production values and an AAA-budget feel to it -- something we arguably succeeded with at the cost of our health and free time. Our "gluttony" instead pertains to the parts of the game that fell outside of the core game experience -- this includes the in-game cutscenes, split-screen, online multiplayer, and the boss fights.

These are areas of the game that we really did not have time to implement to the same level as the games core features. If we had dropped one or two of them at an earlier stage, it would have saved us a tremendous amount of time, and the remaining features would have had a chance to reach a much higher level of quality.

Alternatively, we could have used that time to implement some systems that were planned to be a part of the core experience -- such as a checkpoint system within each level, so that players wouldn't have to restart the entire mission after failing, or the "skip in-game cutscene" feature, which we were forced to remove very late in development due to the amount of bugs it caused.

4. Ignorance: Not Always Bliss

If we look at player behaviors for *Renegade Ops* today, the three and four player online modes have been played to such a little extent that I get a headache just thinking about how low the return has been on that particular investment.

So, how could we have been aware of this before the game was released? Well, it is funny how these types of things usually are hiding right there in front of you, in plain sight. For all our attention to gut feeling and gamer intuition, we somehow completely missed the fact that no one on the team -- or even the company for that matter -- ever really preferred playing the three and four player modes.

They were fun for very short bursts of play, but the game simply got too chaotic with that amount of players. This is not something that applies to all games, but it certainly was the case with *Renegade Ops* -- and we failed to pick up on it as we sat there groaning to each other, "Oh, no! We have to test this in four-player mode, too? Snore!"

5. Vanity: The Failure to Accept Limitations

Renegade Ops was a project without any margin for error from the very start. The initial plan we committed to gave us a total of 16 months with one month for the alpha, beta, and release phases, respectively. Our intuition warned us that this would not be enough, but we went ahead with the notion that we could shift things around once we got to the main production phase. Once we did -- seven months later -- we were very happy and proud with the quality and level of completion of our vertical slice, but the plan still had very little room to shift things around.

In the end, the project was delayed by two months. This had several negative effects, the most important one being that we missed a very important promotional event (by just two or three weeks!) that would have given us major backing in the form of marketing and general exposure. It also lead to a constant feeling throughout the project that we were playing catch-up, and never really got to stop and take the time to reflect on what was going on. Our reflections instead came all at once, after the game was released.

So, what could we have done to stop this from happening? Well first of all, we will never plan a project to have as little as one month, respectively, for each of those last three phases of development. It's just not realistic, and we've experienced that firsthand now.

Second, the next time we feel like a project is stuck playing catch-up with itself, without any signs of pit stops for reflections and perspective -- then it's time to pull the emergency brake and take a look at what's really going on. Perhaps there is a problem with the processes, or maybe it is time to get the axe out for some serious re-scoping. Whatever solution is deemed applicable, as long as something is changed to break that cycle.

Summary

Renegade Ops has been an amazing project to be a part of! Through the highs and lows of its development we have laughed, danced, cried, not slept that much, and thrown hundreds of tiny post-it-paper-airplanes at each other.

The following quote from Giant Bomb's review of *Renegade Ops* sums everything up nicely: "It's big, loud, sort of dumb, and a ton of fun..."

Thanks for reading!

Return to the full version of this article