Dev Blog #3: Teaching an old engine some new tricks

Dec. 7, 2018, 4:29 a.m. Frederik Schreiber

Headshots, enemy patrols, autosaves, moving props and so on.. The choice of using the Build engine is not an easy one. Aside from the asset and engine limitations, one of the biggest things that sets an old engine apart from newer lies in its gameplay quirks.

On a modern engine you can expect to have a powerful scripting language that is tailored for the engine. But did you know that it was actually Duke Nukem 3D which introduced this concept to FPS games with it's "CON" scripting language?

Ion Maiden uses the popular EDuke32 port, a continuation of the lesser known old DOS EDuke.
With the last commercial Build title WW II GI, they needed some additional CON flexibility that Duke3D didn't support. They later released these changes for the original Duke3D as "EDuke".
Despite over a decade's worth of polish, bug fixes and additions, it's still able to execute even the original 1996 Duke code as-is!

By doing things the old way, we can guarantee that everything we do stays very true to what was possible at the time. This doesn't mean that we should discard over 20 years of some really nice improvements to the genre, teaching this old dog to do some unexpected new tricks is part of the fun. This time we'll take a look at some of the new additions which could be easily taken for granted today.

Headshots

A staple of any modern shooter, popularized and cemented to FPS games by the late 90's.
Aiming in Build you say?
You simply pointed at the right direction and the game did the hard job of aiming at the enemy for you, regardless how much above or below the crosshair it was. This was done partly to allow keyboard only play (the standard at the time), and was kept enabled even when players were using mouse control.
This also means that no attention was put on regional damage as it was impossible for the player to hit anything else than the center of an enemy sprite.

To get around this in Ion Maiden, we take a bit of a different approach. We check the distance at which an enemy was hit compared to where its feet are and scale additional damage, like a gradient. For most enemies we also label the the top 1/8th of the sprite as the "head" region and have additional effects that may happen internally, whether it's a different sound, animation, or any number of other things, which may even be invisible to the player but helps smooth out the gameplay.

We still have the silly auto aiming to deal with. Due to code improvements, we can allow it to be disabled completely in Ion Maiden as an option. But, we also came up with a way to allow the old system to work in tandem with headshots.

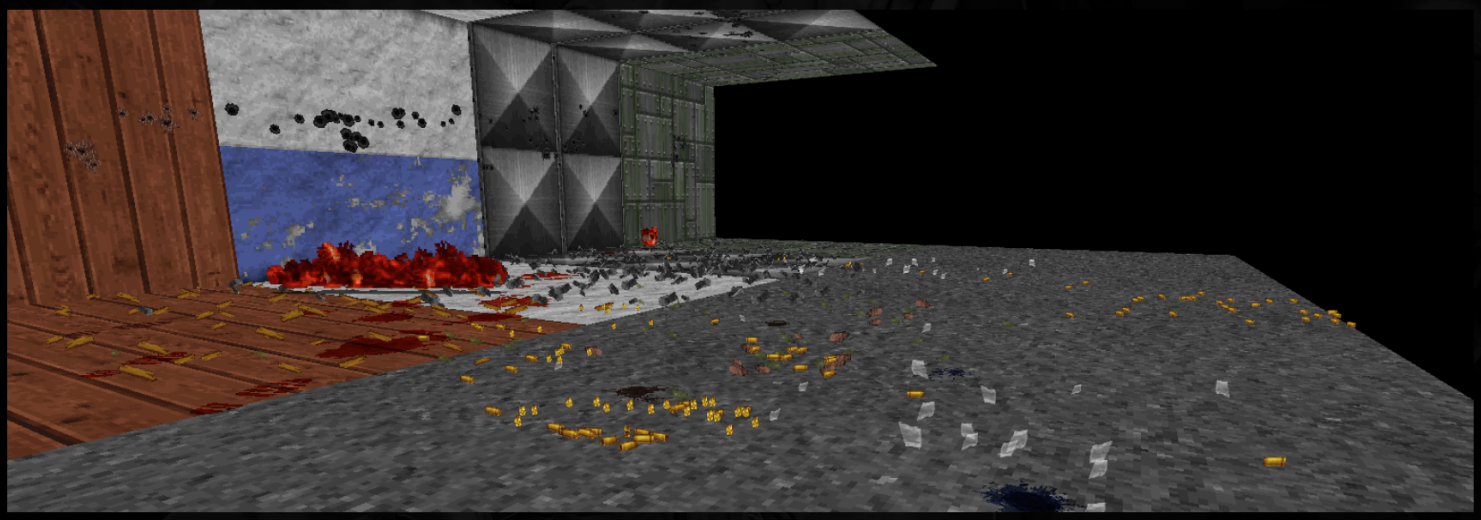
When the player's crosshair is aimed directly at an enemy sprite the game temporarily bypasses the auto-aim system (if enabled), allowing the shot to travel where they are pointing! This lets players maintain that classic feel - but with the enhancement of additional control - improving game feel. It's small enhancements like these that help modernize some of the older parts, while introducing further skill/reward factor.

Particles

Early shooters had lots of destruction, blood and gore. In Ion Maiden, with more powerful computers we took this concept further by allowing you to shoot pretty much any surface around and have scraps be left behind. However there is one big technical limitation: You can only have 16,384 sprites existing in a running map, this includes all elements from the tiniest visual particle to an enemy you fight. Not to mention that the engine would already slow down way before that figure.

We work around this limitation without overwhelming the sprite count by first setting a hard limit on certain kinds of particles. They get added into a queue, and when the limit is reached old particles (including spent cartridges, blood, etc) are removed and new ones can be added. This basic idea actually was present in other Build titles including Duke Nukem 3D, however we have some additional layers.

When a particle would be too small to be seen we remove it immediately so it will never even be added to the queue, and this applies even to temporary particles like sparks which helps improve performance. Additionally when particles are below a certain size (too small to be noticeable long term) we remove it from the map after it has a chance to complete its initial animations.



By doing these things we can keep a full and interesting appearance during all of the on screen action, while doing our best to keep framerates up and avoid hitting the limit for sprites.

Movable props

A mid-90's FPS game generally consisted of very limited movement for objects, your main concern was to shoot anything that moved. Some games had some small things such as pushing stuff based on explosions. Physics were generally nonexistent.

One of our early goals was to introduce movable props in to the maps and it took quite a bit of work here, since this engine was done in a time where no physics engines even existed.

Our extensive prior knowledge having worked with the engine was put to good use, and with experimenting we ended up with a simple rigid body physics system which we could apply to objects at will.

When our objects are given the option to be movable, they are able to check for collisions with not just explosions but also things like weapon fire, enemies, bowling bombs, the player and even other movable objects.

Fast forward a few months and we came up with an idea for a secret that relied on a chair to be moved and soon it was clear that we really needed a way for the player to gently pull objects. This addition really helped players with reaching some of the secrets. We really want to reward and embrace exploration instead of punishing the player with clip walls.

These physics create interesting new situations and interactivity. Even the simple act of knocking a mug off the counter and seeing it shatter makes the world simply feel more alive.

Effect system improvements

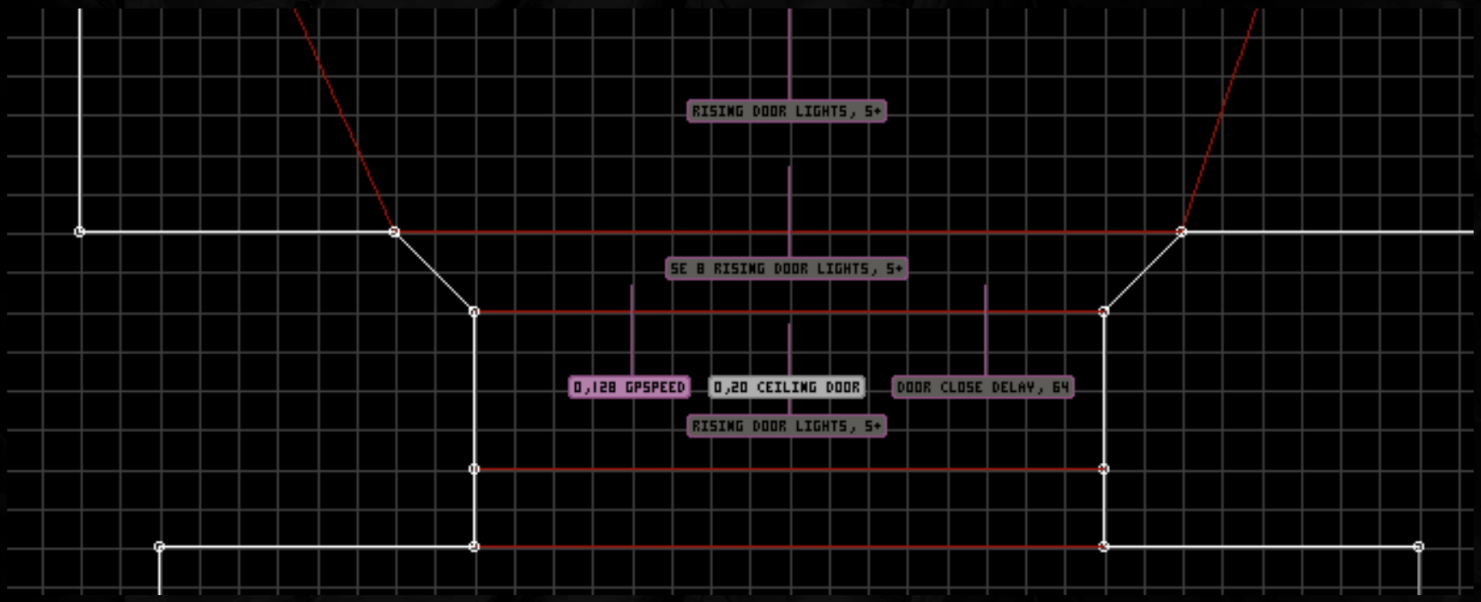
Now off to some more technical bits...
One detail that set Build titles apart was their "effects" (think doors, earthquakes, lighting etc..) and how to trigger those.
This greatly influenced mapping possibilities.

Doing effects in Build games really comes down to utilizing two tags: Lotag and Hitag.
This duo generally consists of an effect number and an arbitrary pairing number for a triggering.

A trigger could be any button, switches, having the player “use” it or have player walk on it.

These two tags could be assigned to all key map elements: Sectors, Sprites and Walls.
Despite looking similar on the surface, almost every Build game developer had to code their own doors, elevators and lighting effects. Tagging system principles, possibilities and numbering can vary wildly between games.

This system was already quite flexible for its time but it didn’t really scale quite well. At first this might seem a lot but once you build a door with a basic lock, you realize that you still haven’t set the speed / sounds / auto-close timers and whatnot. Many games started adding things like special sprites to further define behavior. Some games such as Powerslave (Exhumed in Europe) went its own way and repeated the effect numbers so that every additional 1000 ended up being a speed multiplier. Honorary mention goes to Monolith’s Blood, which had the most flexible system out of them all and ended up completely writing a custom trigger system. It’s a huge task, knowing the shortcomings of existing systems we had something better in mind.



For Ion Maiden, we would need to find a balance that allowed us to have more flexibility for effects and allow us to trigger more complicated sequences without resorting to some very extreme workarounds. Taking a cue from Shadow Warrior, we had some tricks we could use.
There was a third “extra” field which was left as sort of a scratchpad value only for coders to use, while this helped a lot in-game, it was always initialized to a default value of -1 during map creation and only took space. For any sprite that was invisible in-game, we could also utilize its palette index for another 8-bit value, something which was already used in a few games.

But we wanted more!

We went on and hijacked even scarier internal values which, like “extra”, were only utilized in-game but this time they were much more error prone due to being internal movement velocity and owner values. Now during map load there is a new pass which will quickly restore safe defaults before the map is fully loaded, while reading the values in to script memory. Suddenly we gain 4 new additional tags (xvel yvel zvel owner) that can be exposed for mapping!

The goal was to give mappers very extensive control over parameters and to make each effect as modular as possible, including optional parameters that weren’t specifically requested.

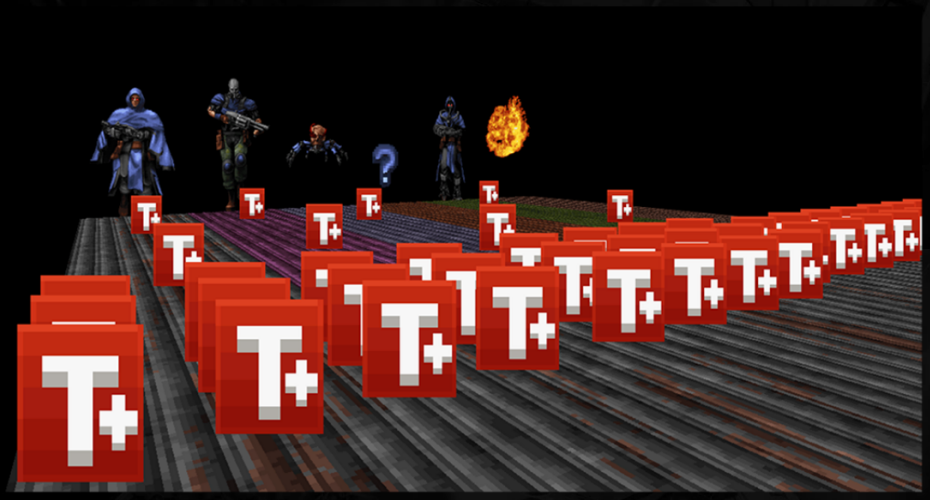
Now there is much less reliance on additional “information” sprites for effect attributes, these can now be nicely set from one place. Documentation is also now much easier since even more complicated and multi-purpose functions can be done by one effect sprite alone.

```
TOUCHPLATE+ (TOUCHPLATE WITH STYLE? ACTIVATES ALL KINDS OF THINGS?)

WILL ALSO TRIGGER IF PLACED INSIDE A CEILING OR FLOOR DOOR WHICH CLOSES
HITAG (0) = [STARTING] LINKING TAG
LOTAG (0) = NUMBER OF ADDITIONAL TAGS TO ACTIVATE
EXTRA (-1) = DELAY BETWEEN TRIGGERS IF MULTIPLE TAGS
ZVEL (0) = OTHER TP+ SPRITES SHARING THIS TAG WILL DELETE WHEN THIS ACTIVATES
OWNER (-1) = LINKING TAG TO ACTIVATE THIS TP+ WITHOUT PLAYER/SECTOR INTERACTION
PAL (0) = VARIABLE USAGE (0 or 9 toggle) [OFF]
    | IF IN DOOR = RESET ON OPEN WITHOUT RETRIGGER | ELSE ALLOW REACTIVATION ON SECTOR RE-ENTRY

XVEL (0) = FLAGS (PRESS M TO LIST)
    | [1] TRIGGER LOTAG MULTI-ACTIVATION SEQUENCE COUNTING BACKWARD FROM HITAG
    | [2] FOR MULTI-ACTIVATION. LOOP ACTIVATION CYCLE. USE WITH PAL 9 AT YOUR OWN RISK.
    | [4] IF PLACED IN A DOOR AND DOOR IS CLOSED DON'T TRIGGER UNTIL DOOR STATE CHANGES
```

In-editor documentation for Touchplate+, the go-to trigger, showing all the options.
Despite the amount of options, in its simplest form you only need to set it a Hitag.

With Touchplate+ and it’s additional tags, we also tackled the issue of triggering multiple things at the same time because now we don’t have to choose between using precious tags for a parameter or pairing, we could always have both.
With TP+ we can easily have sequences of effects, set a fixed delay between them and auto step between tags, and even link to other TP+ sprites, allowing some extremely elaborate sequences to be created by just placing multiple TP+ sprites.

Our “Horde game mode” Queen of the hill started as a brainstorming session between two developers and without any additional code, a single round prototype was cooked up in a day by using only existing triggers and effects. With some supporting code such as HUD and kill counters, all the rest was done with nested TP+ chains and reusable respawn sprites.



Enemy patrols and squads

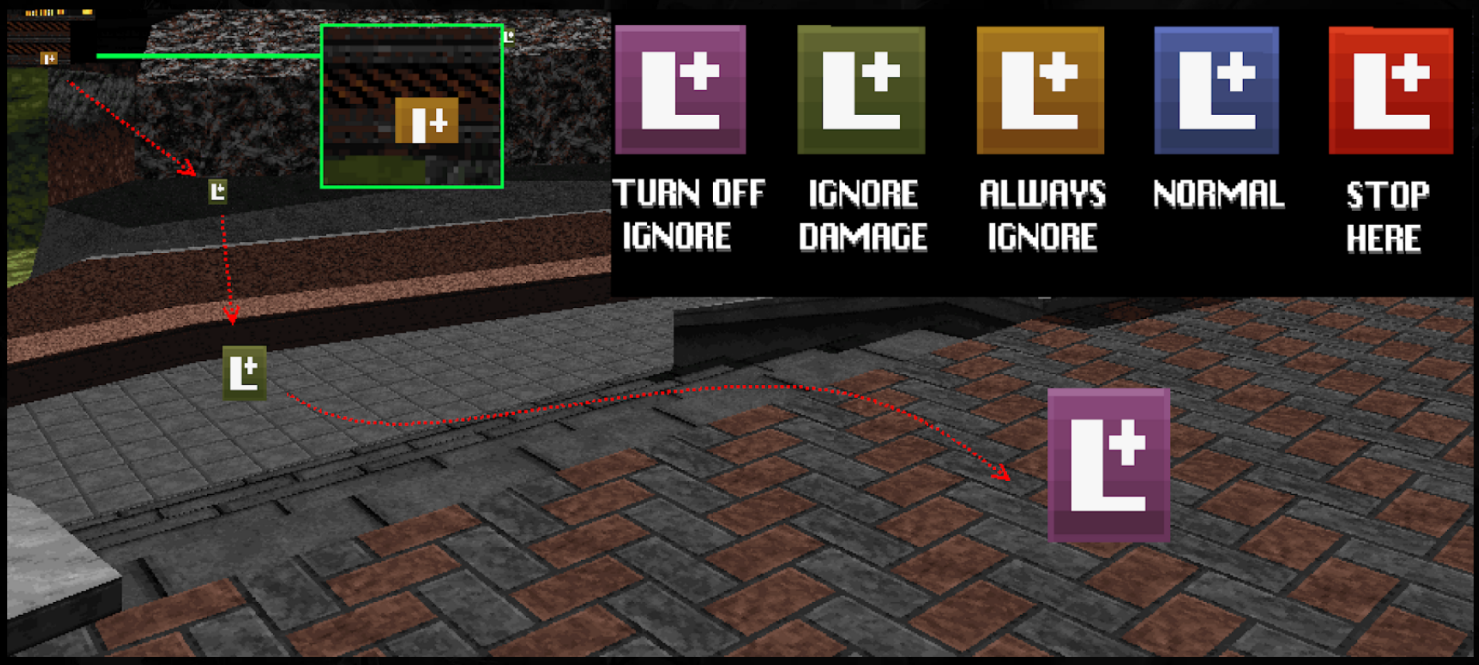
One big challenge with creating interesting encounters on older games is usually the A.I.
You set up a cool ambush, hole gets blown to a wall and finally when it’s action time, the enemies just decide to roam elsewhere instead.

One could build an advanced, complicated A.I. but that would just kill the fun if enemies would suddenly start ducking for cover in a 90’s FPS. We needed a balance.

To get around this, we started experimenting on a new path system by expanding on existing special sprites called “Locators”, something familiar to many Duke mappers but this time around they do much more than before.
At first we wanted to spice up areas with simple roaming patrols. This allowed us to have some variety in more open areas without having 10 enemies visible far away and standing like statues.

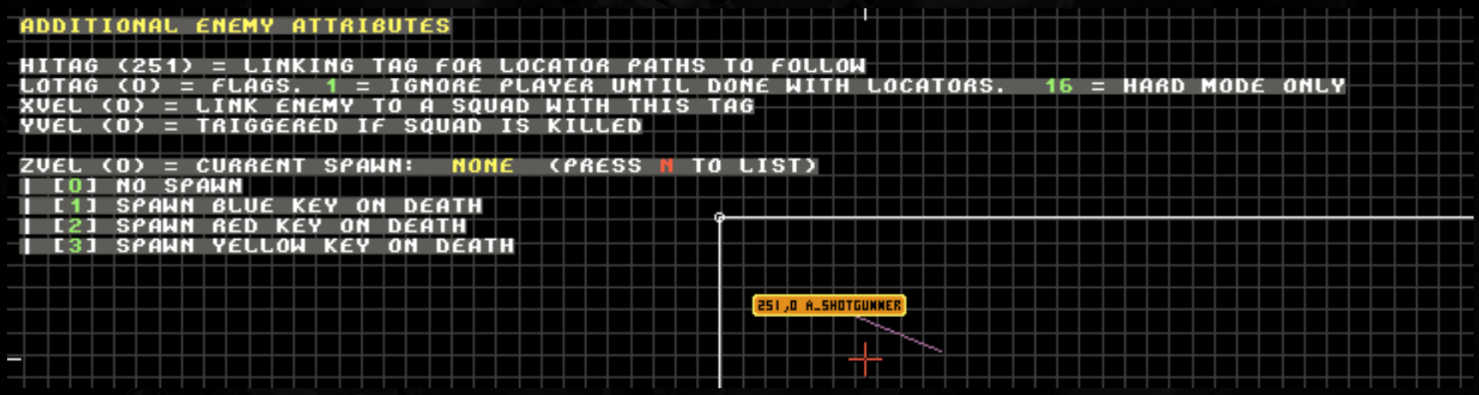
Naturally I always end up bugging our coder about more additions that we could use.
This resulted in various behavior modifiers where we now have almost full control over the initial paths of enemies and are able to set various A.I. modifiers.

By using this system you can create all kinds of situations, from making a group of enemies walk around a corner as you approach, to even complex systems such as Queen of the Hill which extensively uses these programmed paths to funnel in enemy waves.



Proper use of ignores prevent enemies from getting stuck at spawn points, but also gives some breathing room during more intense waves.

But we didn't stop there. Now there is also the ability to tag enemies as being part of a group (or "squads" as well call them). The mapper then is able to assign either a keycard spawn, a trigger (for example, a door opening or a ceiling collapsing), or both to this group. When every member of this squad is defeated the corresponding action will occur.



Mappers can use this not only to gate progress, but also to create interesting consequences for choosing to fight enemies or avoiding encounters. If you have played our Preview Campaign you can see an example - entering a metal detector will cause you to be temporarily sealed in until you eliminate the enemies that appear.

Other additions

Hubmaps

Similar to level transition seen in Half-Life, these allow us to split areas between multiple physical map files. This not only saves performance but also allows us to have huge highly detailed maps as there are physical limitations with how many objects a map can hold.
More importantly, the player can always return to previous maps and everything would be exactly as he left it. This also allows backtracking or even other paths that connect back to previous levels.

Autosaves

Many gamers probably remember having their finger hovering over F2 or F5, autosaves and checkpoints were a foreign concept for a long time. You were expected to manage saves on your own, that's not really fun. Adding autosaves did pose some challenges as the game only had 10 save slots and had no concept of this, not to mention that players might still want to do manual saves. One especially tricky part were quicksaves, which would default to last used slot, and would overwrite these autosaves. In the end the whole system was overhauled to work without fixed slots and to allow different special slots that are internal to things such as map transitions and checkpoints.

GTFO

One downside with the Build engine is that the collision detection can get quite interesting at times. Even with a blocked wall or a window you can accidentally (or intentionally..) push yourself through geometry with enough speed. A lot of the time this can give quite undesirable results such as teleporting the player into completely different areas that occupy the same space.
We don't want to kill all the fun, so we only use this sprite to push the player away from things like narrow ledges on window frames which are meant to be fully blocked but which the engine doesn't really know how to handle due to poor precision.

Secrets

We love secrets, unfortunately the granularity of tagging a secret was normally limited to sectors, which meant that if we had stuff hanging above, it was nearly impossible to tag those as secrets properly. Not to mention that sector tags were also used for other things such as marking areas as underwater, meaning you couldn't have both. Now secrets can be not just classic step-on sectors but also individual pickups can be tagged and even triggered on demand, such as waiting for the result of an explosion and playing out the jingle once the new path is visible.

Z-kill

Last but not least, bottomless pits are classic traps. Typically these got made as extremely long drops that was guaranteed to kill the player due to fall damage, which some times didn't always work due to engine quirks. This meant that we should somehow kill off the player proper so that he wouldn't be stuck at the bottom. But instead of just that, we added a bit of late night humor in the mix: when you're about to fall in to one of these, time slows down a bit so that you can reflect at your mistake with a prompt "You're about to fall to your death - Press use to scream!".

There are many more new features and effects not presented here. Build has a lot of potential but very few games exposed that, his is why our motivation was to make Ion Maiden as flexible as possible for mappers. We hope that our tools will inspire people to make some crazy things and stuff we didn't even think of!

Again, thanks for making it this far!
If you haven't played Ion Maiden yet, consider checking it out! *wink wink*

On behalf of the Ion Maiden team, I'd like to thank for the overwhelming support we've received and we hope to return with some more behind the scenes stuff later on!

Signing off,
- Max 'oasiz' Ylitalo / Level design lead
& Jonathan 'Mblackwell' Strander / Game code lead

Tags: ion maiden | voidpoint | dev-blog

BACK TO TOP

SOCIAL MEDIA

Do you want to talk with us about something? Get in touch below!

Contact us about your game at pitch@3drealms.com

RECENT NEWS

Aug. 28, 2023, 1:05 p.m.
The Tempest Rising Demo is now Available!

Aug. 16, 2023, 6:30 a.m.
3D Realms Gamescom 2023 Line-up!

Aug. 11, 2023, 12:57 p.m.
Tempest Rising - Playable Preview OUT NOW