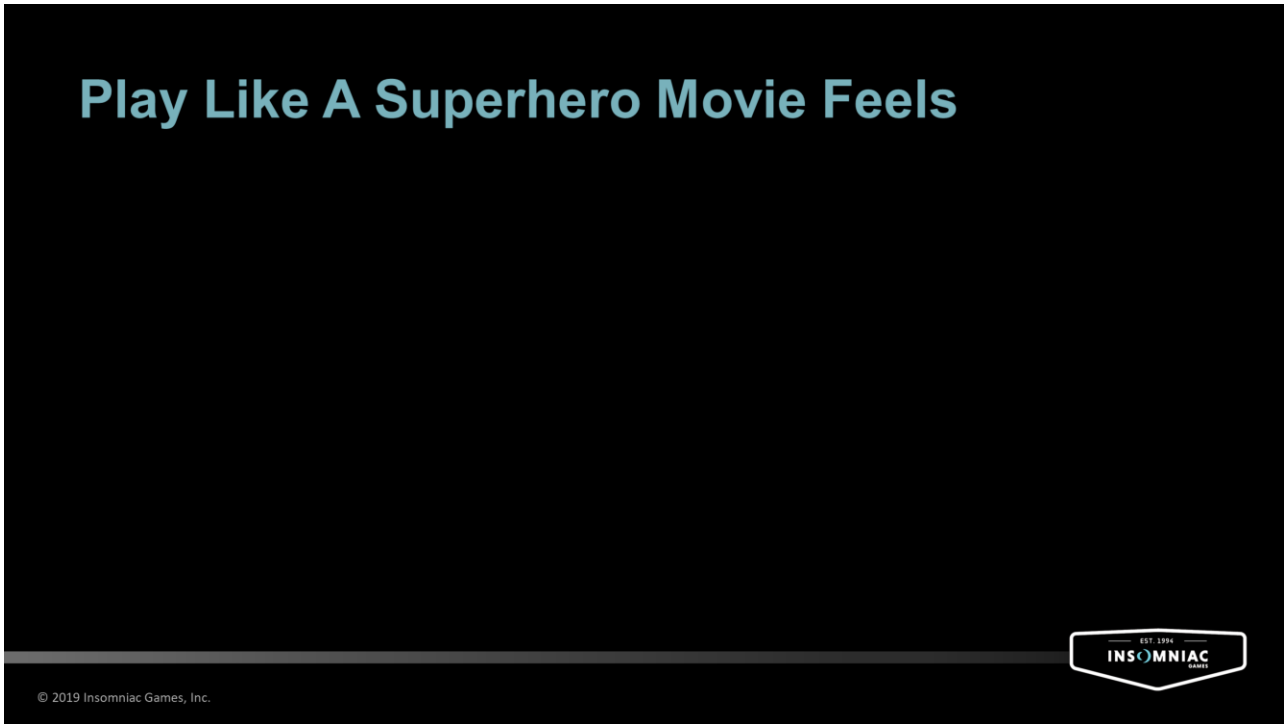

GDC
Concrete Jungle Gym:
Building Traversal in "Marvel's Spider-Man"
Doug Sheahan
Lead Gameplay Programmer
Insomniac Games
GAME DEVELOPERS CONFERENCE
MARCH 18–22, 2019 | #GDC19

- Thank everyone for coming
- Survey and phones
- My name is Doug Sheahan, I'm one of the lead gameplay programmers at Insomniac Games and we're here to talk about building traversal in "Marvel's Spider-Man".
- So, let's get into it

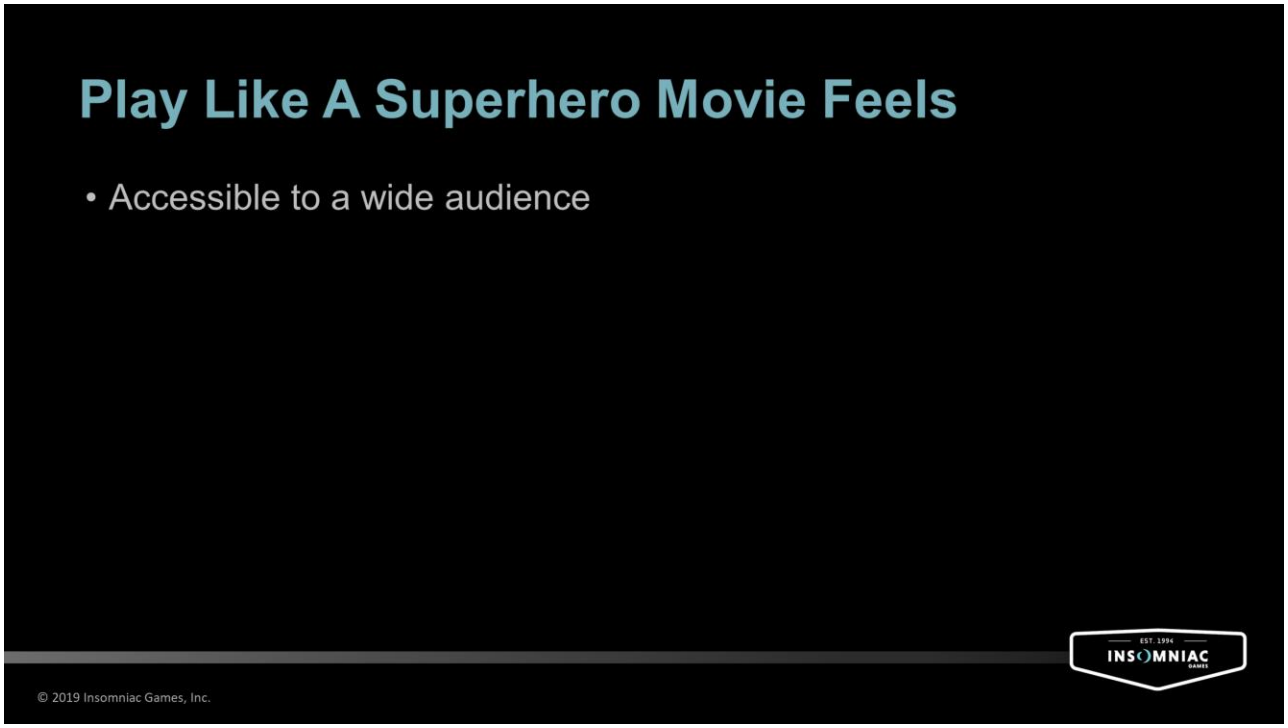
- In the fall of 2014 I found out Insomniac Games was making a Spider-Man game and that I was assigned the task of bringing his traversal to life.
  - This would not only be the biggest game Insomniac had ever made, but it was using a legendary IP

with a heavily invested fanbase and an unrivaled expectation of quality.

- As someone who was once a poor college student holed up in bookstores reading Spider-Man comics, the idea of making Spider-Man swing was super exciting.
- It also scared the hell out of me.
- Fortunately, we had the game's vision statement as a starting point: "Play like a superhero movie feels".
- So we set upon a journey to, as the saying goes, make you feel like Spider-Man.



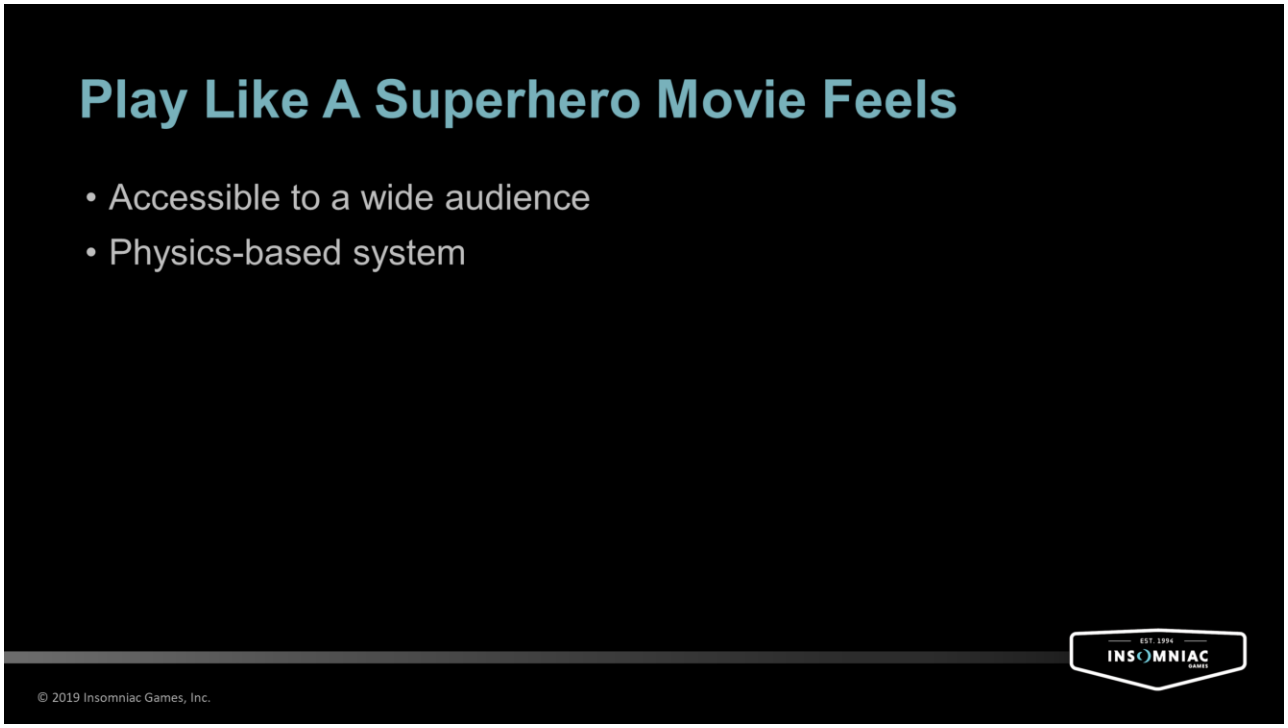
- So what does “Play Like a Superhero Movie Feels” even mean for traversal?
  - With swinging as our starting point, we decided on a few core beliefs that would guide the feature’s development.



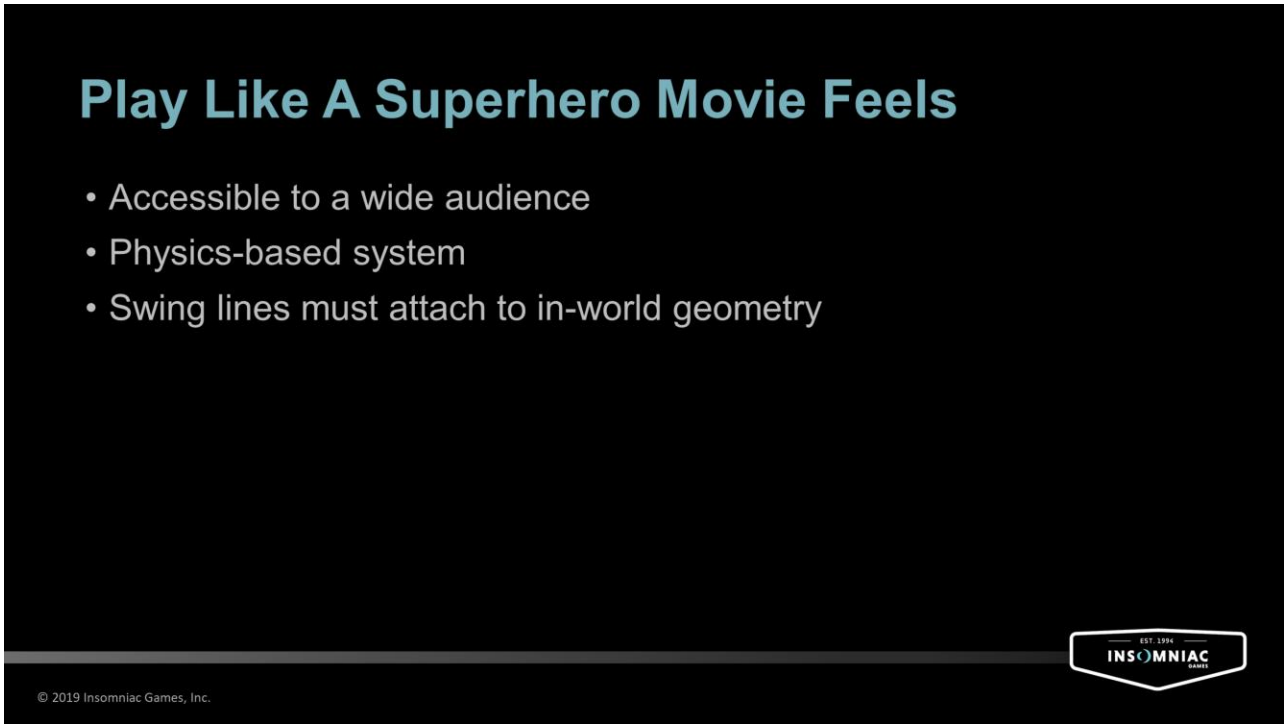
- The first thing we acknowledged was that we wanted to target a broad audience.
  - Spider-Man has a worldwide following and we wanted as many people as possible to be able to play and enjoy the game.
  - However, we knew we had the

issue that most gamers didn't really have any muscle memory for swinging.

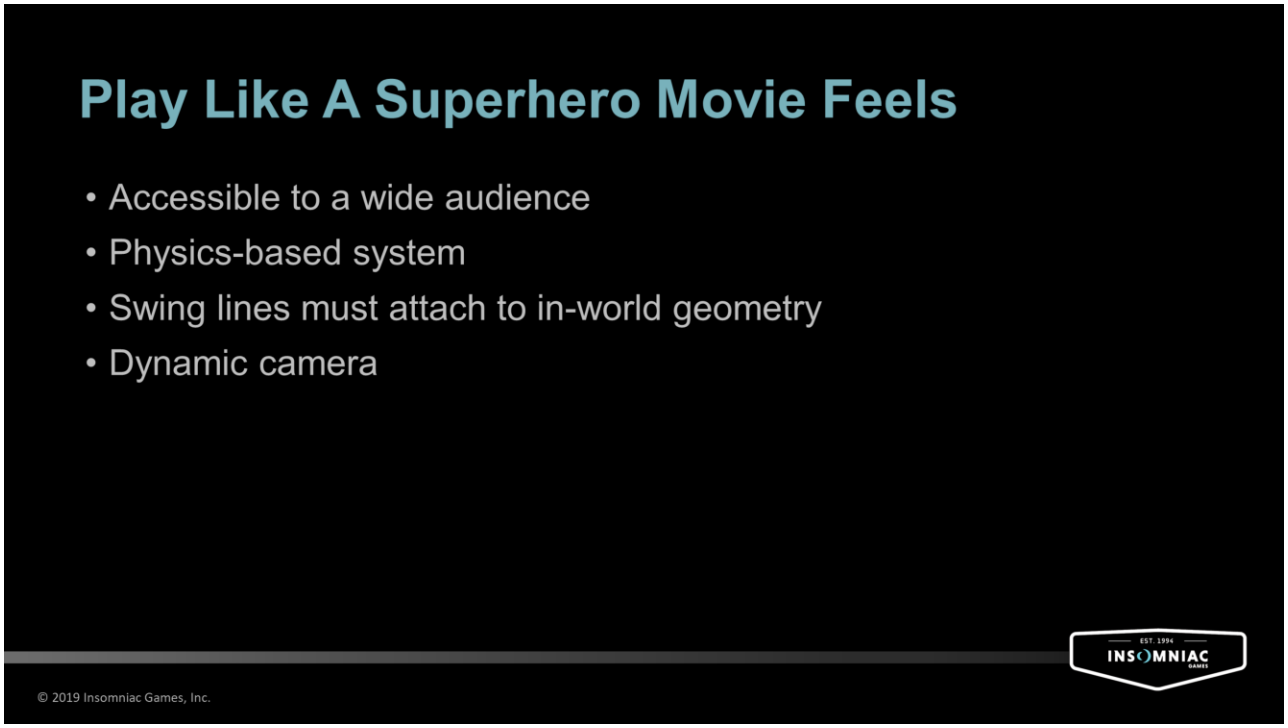
- Unlike, say, a third or first person shooter, most folks don't really have a brain map for how to make swinging go.
- To add to the challenge, we start the game by chucking you out a window, giving you control and saying, "Alright, swing time, go"
- Wanting that first moment to feel just right, we knew we had to get the controls to a point that were easy to learn, but hard to master.



- Next, we wanted traversal, and swinging in particular, to be rooted in physics.
  - This was vital to give players a sense of believability, a feeling that swinging through New York as a superhero was actually possible.

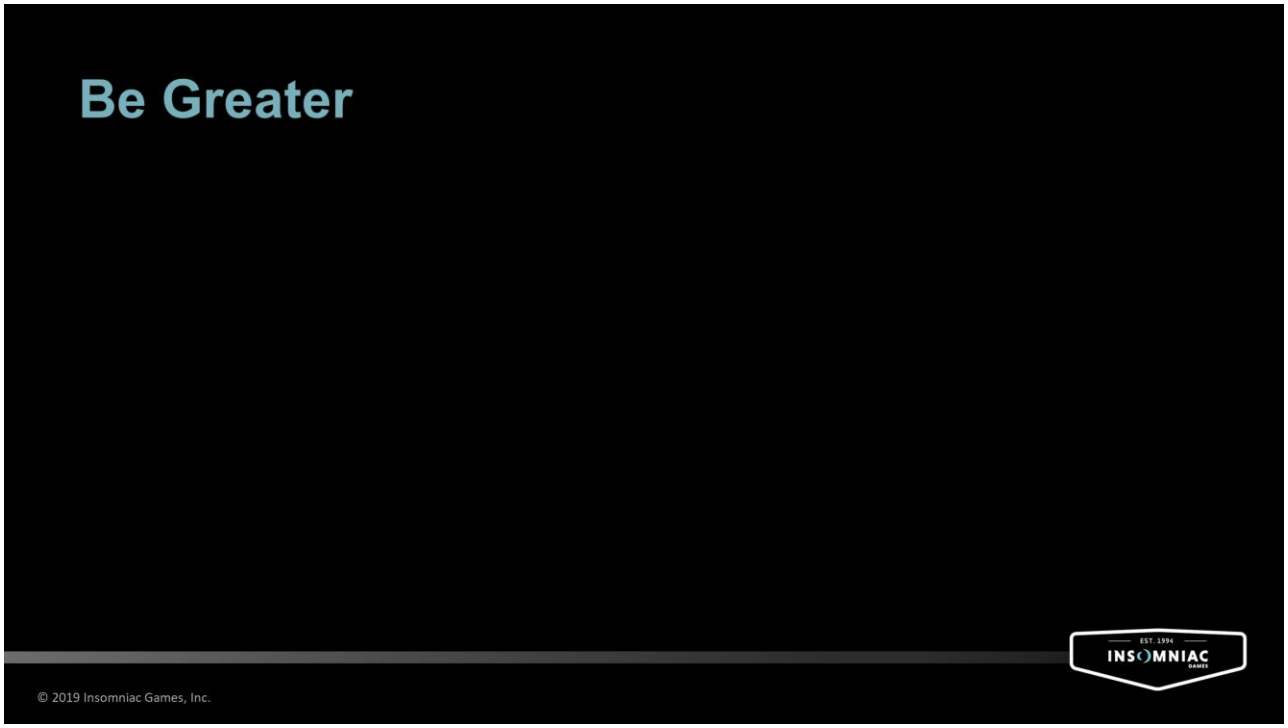


- Following that, swing lines had to attach to in-world geometry.
  - This was extremely important in order to ground the simulation correctly but mostly we were terrified of an angry fans telling us we'd ruined their childhood.

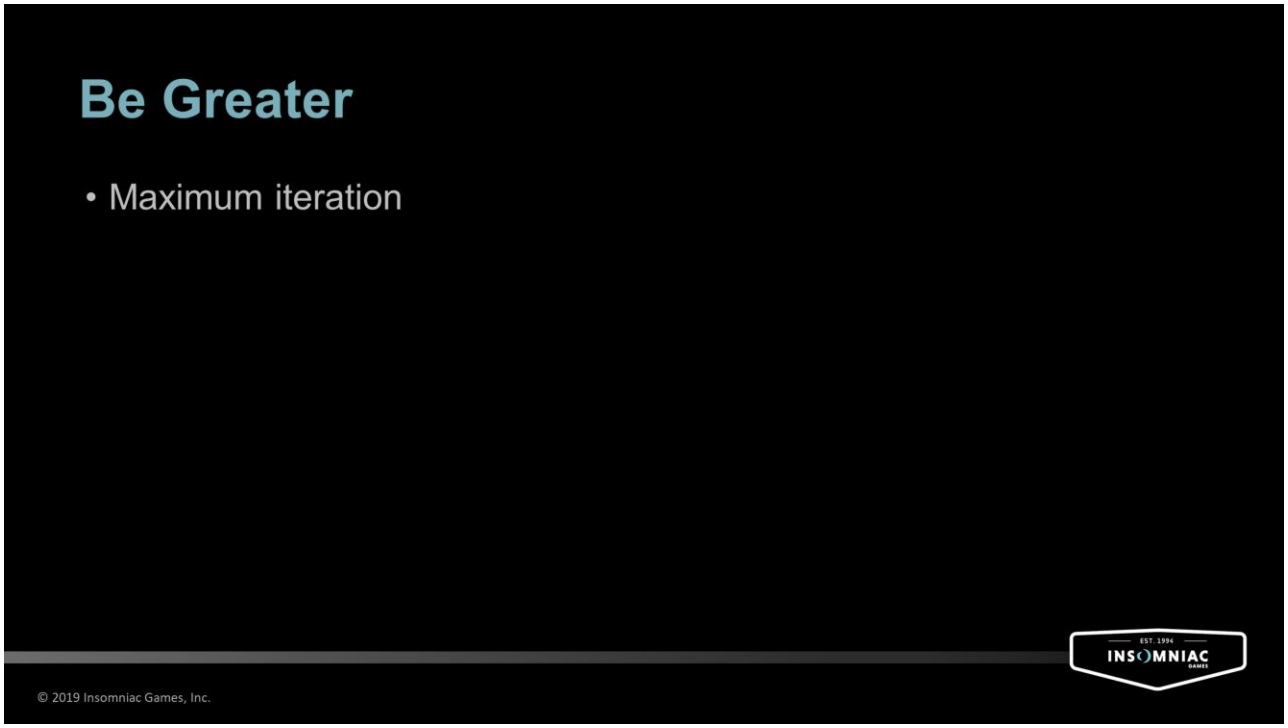


- Finally, we wanted to have a dynamic camera.
  - The look and feel of swinging needed to be fluid, fast, exciting, and cinematic.
  - We needed a camera that helped translate all the high-flying acrobatics and velocity into a

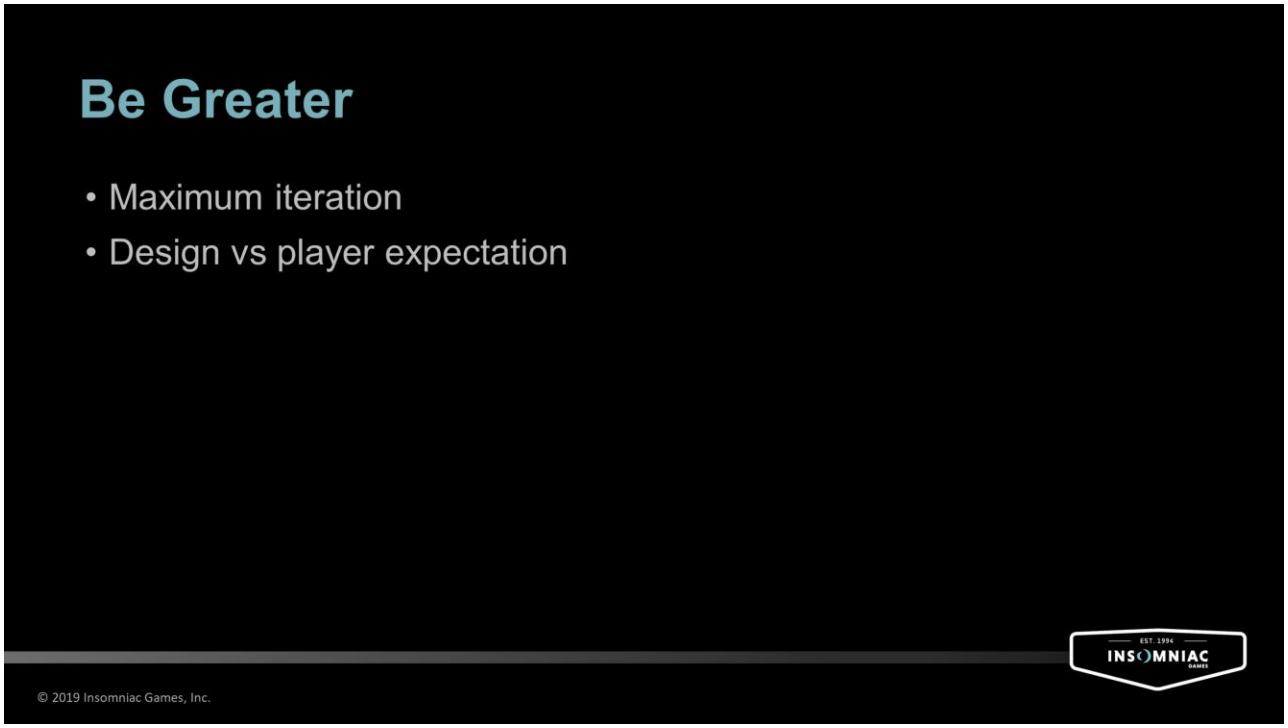
visceral experience for the player



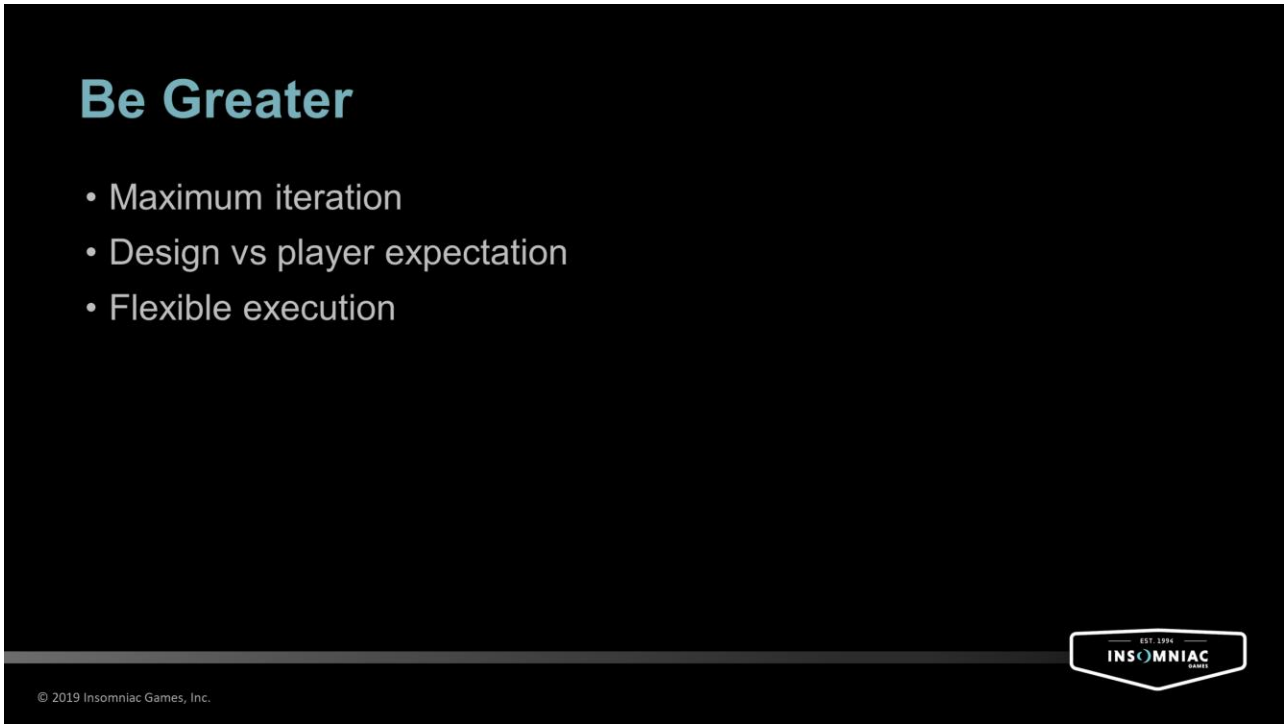
- Today, I’m going to show you how we achieved those core principals by taking you through the process by which we created traversal.



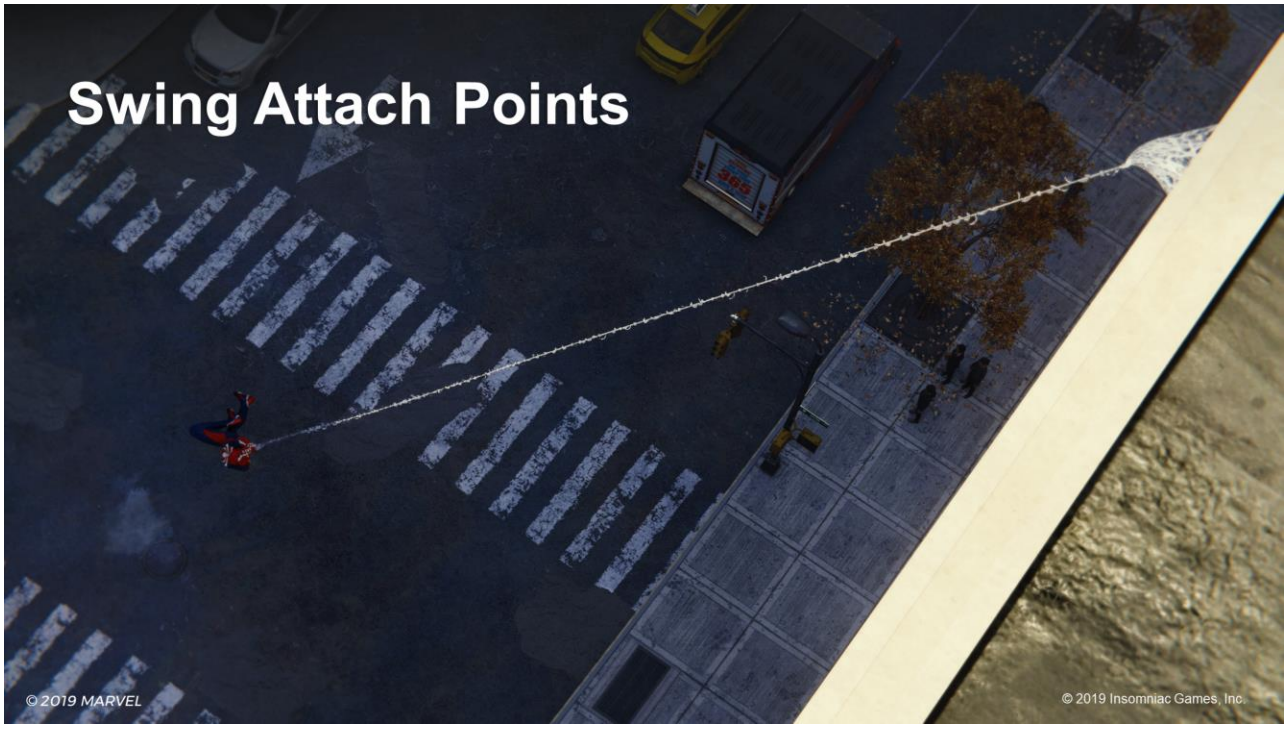
- The biggest key to our success is iteration.
  - For many elements I'll show you not just what we ended up shipping, but also where we started, failed, learned, and improved.



- However, knowing when we needed to improve wasn't always obvious, it came from a steady diet of playtesting and learning how player's interacted with the game and where it failed them.

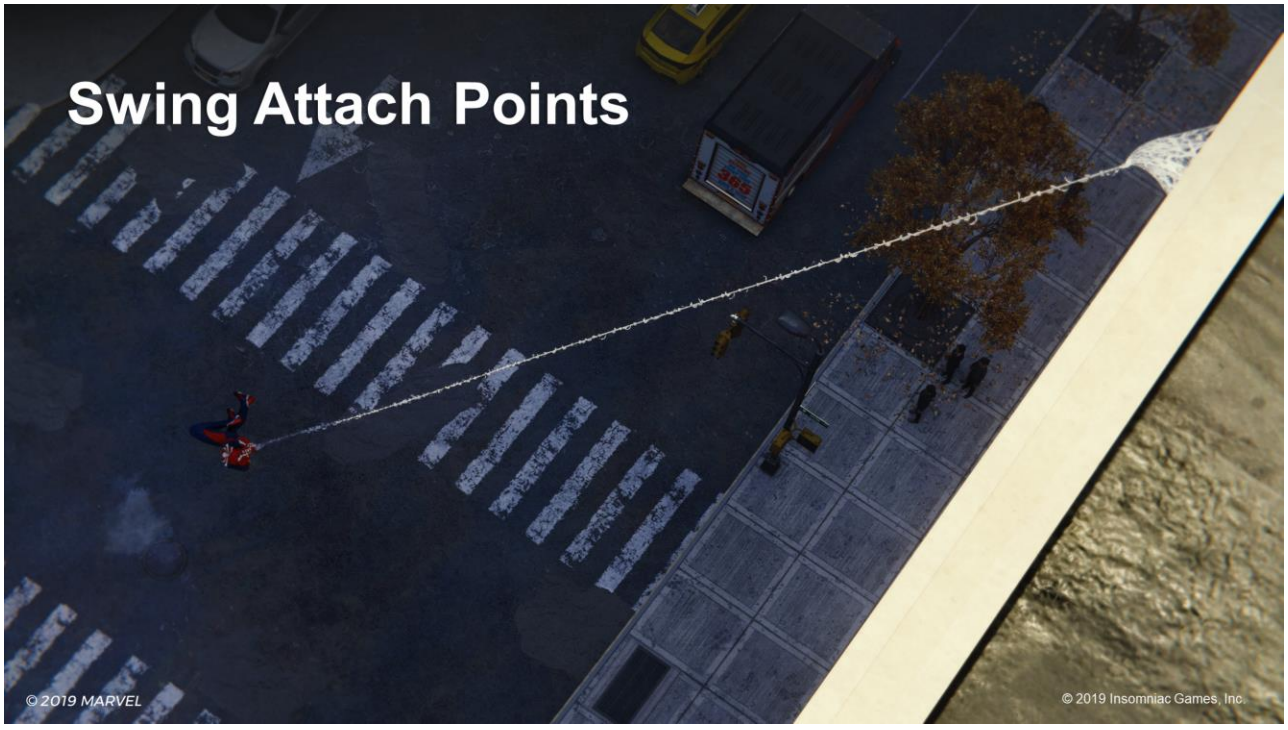


- This meant we had to stay flexible on our execution and always look to be greater.



- Now, before we can swing, we have to have something to attach to!
- As stated, we wanted swinging to be accessible and this meant we need to think about attach points from the player's perspective.
  - Namely, what is the player actually responsible for when it comes to

picking an attach point and what is the system responsible for.



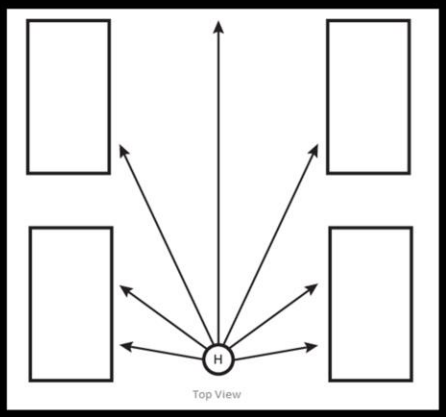
- We first decided that we wanted players thinking more about where they wanted to go rather than where they were attaching.
  - This removed a significant burden from the player and let them focus on additional layers of gameplay like chasing cars and dodging

bullets, rockets, and the occasional pigeon.

- The player would communicate their directional intent with the left stick and the system would take it from there.

Swing Attach Points: Version 1.0

- Array of ray casts



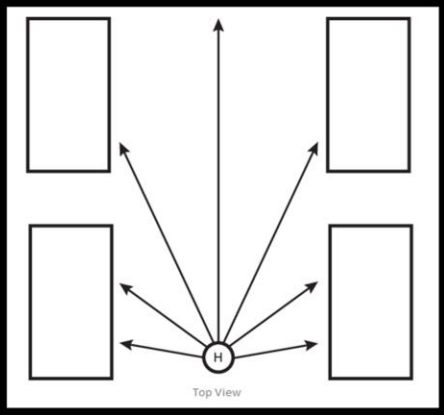
Our first attempt at finding attach points was to comb the environment using an array of ray casts.

- This would minimize the required effort from the design and environment teams
- And ensure that we are finding and attaching to geometry

However, this approach presented a number of problems that we could not overcome…

Swing Attach Points: Version 1.0

- Array of ray casts
- Problems:
  - Not enough resolution

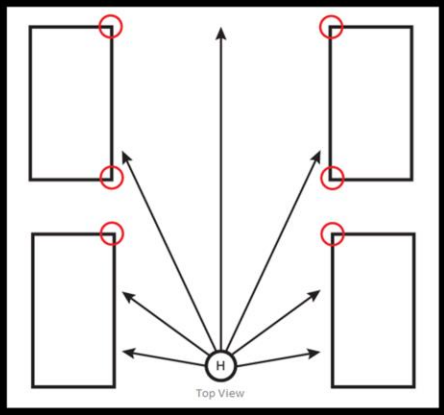


- The biggest issue is that ray casts simply did not provide enough resolution.
  - Our line lengths would often exceed 50m and even with a respectable density of ray casts we were getting 20m square gaps at full range.

- This was enough to not only miss smaller objects we wanted to swing from like flagpoles and radio towers, but completely miss large sections of buildings entirely

Swing Attach Points: Version 1.0

- Array of ray casts
- Problems:
  - Not enough resolution
  - Poor detection of corners



- As an additional issue, we recognized that corners were often optimal swing points but ray-casts were miserable at finding them.

Swing Attach Points: Version 1.0

- Array of ray casts
- Problems:
  - Not enough resolution
  - Poor detection of corners
  - Geometry with no collision

- A more obvious problem is that ray casts can’t hit geometry with no collision.
  - This was particularly problematic with trees in areas like central park
  - Even if those objects did have some small bit of collision we

could swing from, the ray casts were still terrible at finding them.

Swing Attach Points: Version 1.0

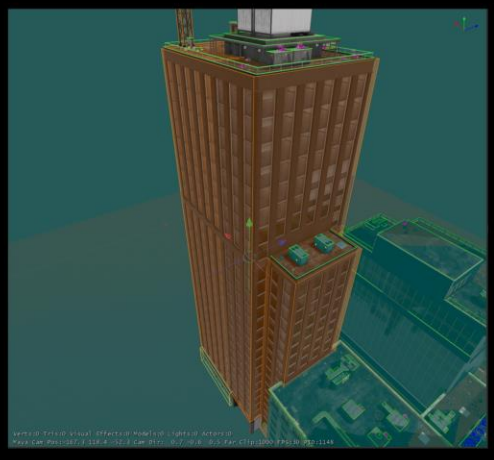
- Array of ray casts
- Problems:
  - Not enough resolution
  - Poor detection of corners
  - Geometry with no collision
  - Post-Adjustment

- Finally, as we'll outline later, many points require additional fixups to hit optimal slopes and lengths.
  - Doing these adjustments via collision checks could prove extremely costly when trying to process dozens of potential attach points and still may not be

accurate.

Swing Attach Points: Version 2.0

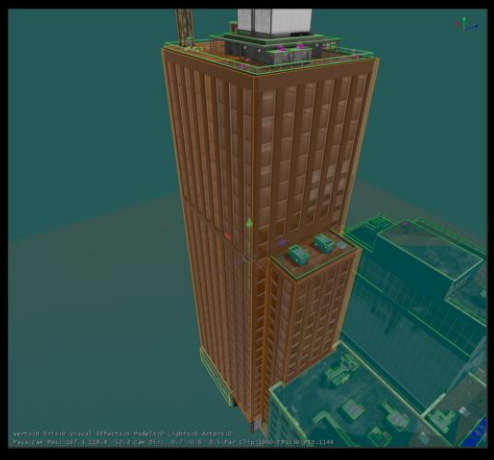
- Box Volume Markup
  - Process faces
  - Ray cast to final point only



- With ray-casts coming up short, we decided to use a markup based approach by wrapping our buildings in box volumes that approximated the shape of the building.
  - We would find nearby volumes and process them on a face-by-face basis
  - Then perform a final raycast only on the best point to get our true attach point.

Swing Attach Points: Version 2.0

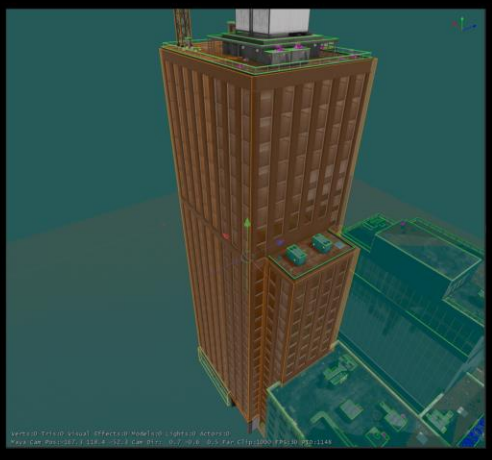
- Markup Advantages



Moving to markup provided a number of advantages that were absent when using ray-casts.

Swing Attach Points: Version 2.0

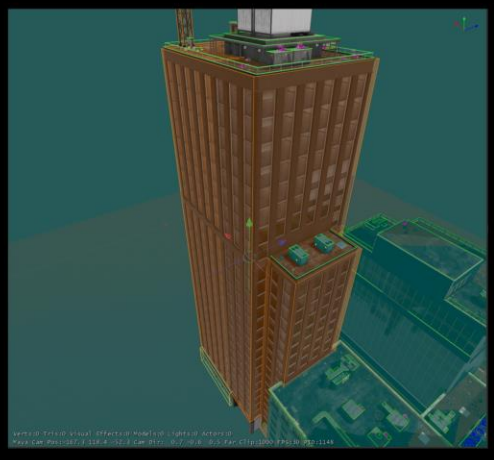
- Markup Advantages
  - Fast point collection



- First, point collection was really fast.
  - Using a cache-friendly sphere database of just our data significantly helps scalability.

Swing Attach Points: Version 2.0

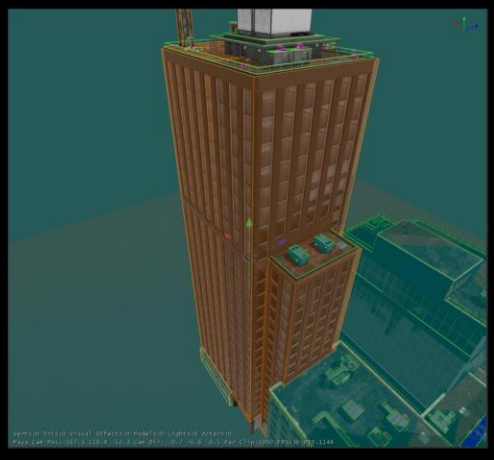
- Markup Advantages
  - Fast point collection
  - Infinite resolution



- By using shape information, we effectively had infinite resolution for point placement on each face of the volume.
  - This allows us to easily find edges and corners and never have to worry about missing smaller objects

Swing Attach Points: Version 2.0

- Markup Advantages
  - Fast point collection
  - Infinite resolution
  - Designer control



- Finally, by using markup we are able to afford designers additional control over various gameplay aspects like custom swing parameters

Swing Attach Points: Version 2.0

- Problem: Scalability

- The main drawback with markup was one of placement scalability.
  - We have a huge world full of things that require markup.
  - Placing all of this by hand and maintaining it as the environment kept changing posed an enormous amount of upkeep and

potential bugs.

- Our solution involved two primary tools:

Swing Attach Points: Version 2.0

- Problem: Scalability
  - Houdini for procedural placement

- First, we had to handle buildings. Fortunately we were using a system called Houdini to help us generate the geometry for our environment out of building blocks.
  - This software could also be leveraged to analyze the geometry and procedurally place volumes to

approximate our building shapes.

Swing Attach Points: Version 2.0

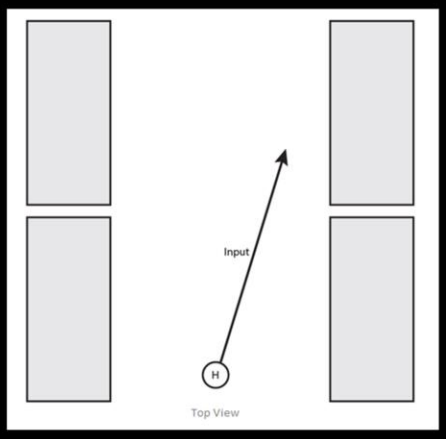
- Problem: Scalability
  - Houdini for procedural placement
  - Prefabs for unique objects

- For objects outside of Houdini's operation, we deployed our engine's prefab system to bundle markup with those objects so that any time an instance was placed it automatically brought the markup with it.

- With our markup now populating the world, we need to actually do something useful with it.
  - At any point the immediate area can be filled with hundreds of volumes that are potential swing points and we need to find the best one.

Picking A Final Attach Point

- Pick a reference point

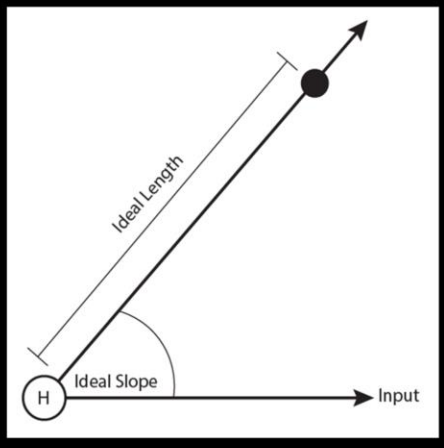


- In order to pick our final attach point, we need to start with a reasonable reference point as our ideal position.
  - Here we have a top-down view of our hero surrounded by buildings.
  - Player influence over point selection comes from input on the left-stick and…

- as stated before, we want the player thinking about where they’re going, not specifically where they are attaching, so the input direction just provides a starting point.

Picking A Final Attach Point

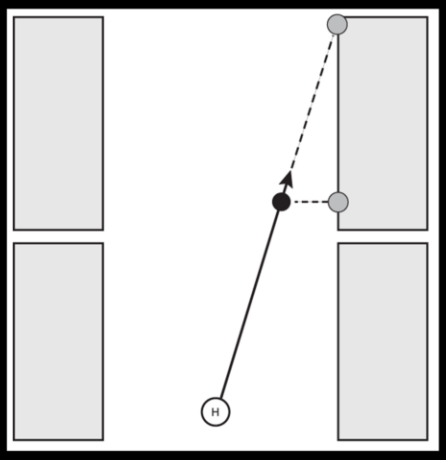
- Calculate ideal swing location



- From that input direction, we apply an ideal slope and line length to transform the input direction into a position in the world that represents our ideal swing attach point.

Picking A Final Attach Point

- Two Position Per-Face
  - Closest Point
  - Ray-Plane Intersect

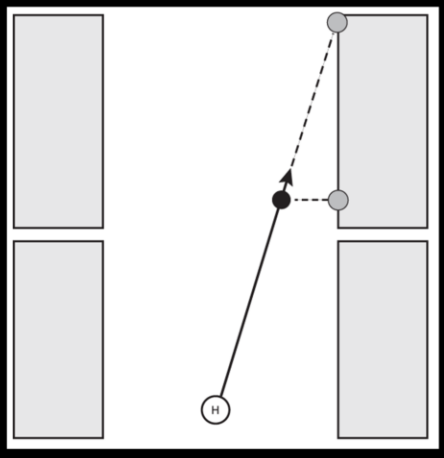


Moving back to a top-down view here, we then use that position and direction to generate two more points on each face

- The point on the plane closest to our reference point and a ray-cast point along our ideal line direction, which is clamped to the volume bounds

Picking A Final Attach Point

- Two Position Per-Face
  - Closest Point
  - Ray-Plane Intersect
- Blend between two based on input versus face normal

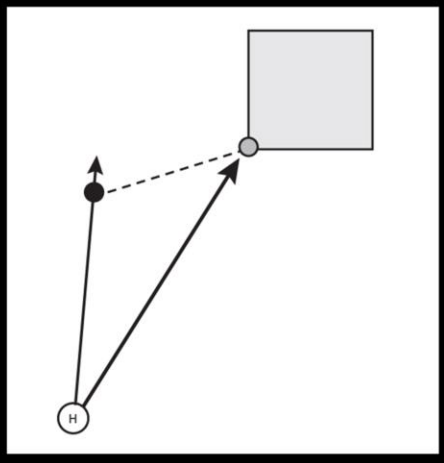


- We can then blend between these two points to get our output point
  - The closest point is useful when we are traveling parallel to the plane
  - The ray-cast point is useful when we are traveling towards the plane

- We blend between the two points based on the difference between the input direction and the plane normal of the face

Slope Refinement

- Closest point not quite ideal

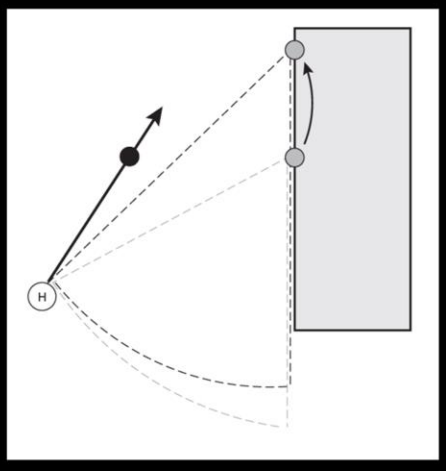


- Now that we have our point, we need to refine it a bit in order to make it more effective.
  - Our blended point can change the slope and line length significantly leading to longer, flatter lines, shorter, steeper lines, and everything in between.

- In order to try and give the player as consistent a swinging rhythm as possible, we want to try and get closer to the ideal.

Slope Refinement

- Shift to improve slope and/or length closer to ideal



- This is an example of shifting the attach point up a bit in order to get closer to the ideal slope which had gotten overly flattened by the closest point.
  - By moving more towards the ideal slope we prevent the player from an excessive drop during the swing,

keeping things more consistent for the player.

Scoring Points

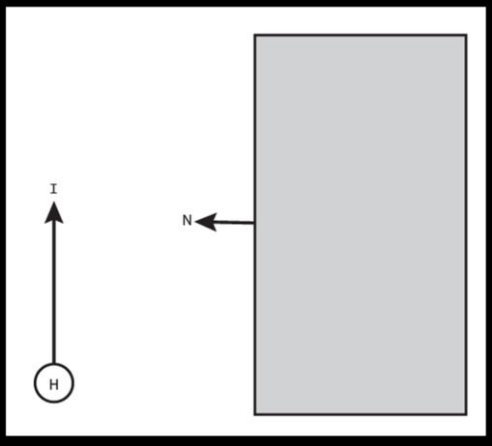
- Use scoring algorithm to choose between dozens of potential points

- Now that we've generated a field of points across the various markup faces, we need to actually decide which one will be our swing point. We do this by scoring each of the points and taking the best one.
- Scores are geared to fulfill the desire of translating stick input to a point

that moves us in that direction.

Scoring Points: Face Normal

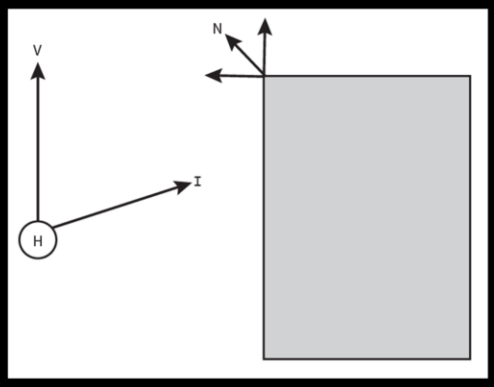
- Higher score for faces that move us in desired direction
- S = 1 – Dot(I, N)



- Our first scoring element is how the face normal relates to the input direction.
- We are looking for normals that are perpendicular to the input direction as this means that the player will be able to swing along that surface in the direction they desire.

Scoring Points: Corner Normals

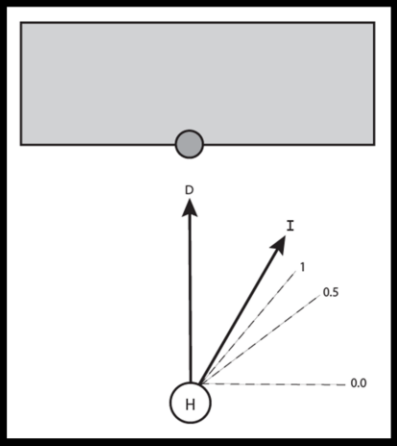
- Snap to corners when close
- Take best of swept normal around corner
- Bonus added to score for corners that help us turn



- Because corners provide players the most control, we try and snap attach points to edges when nearby.
  - We can then use the best normal available to input for our score calculation.
- We can also give corners a bonus when trying to turn.

Scoring Points: Input Direction

- Score on angle between input, direction to point
- Keyed linear delta score



When taking input into account we:

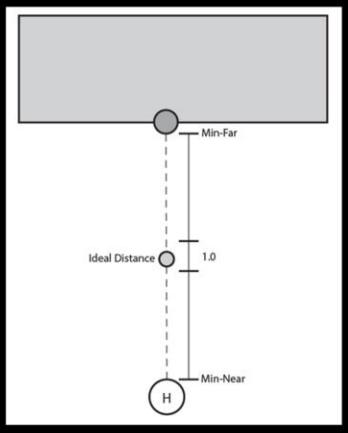
- Start with the angle between the input direction and the direction to the attach point and score the delta

[BONUS INFO]

- We can also dynamically adjust the score when we want to turn
  - When turning, we don’t care as much about precise accuracy, just that the attach point is in the right general direction.
- Using a angle-to-score curve improves over our original implementation using a dot-product because the delta resolution for small angles can be better controlled.
    - This becomes important for long lines where small angular differences can have a big impact on final swing result.

Scoring Points: Range

- Ideal distance is baseline
- Parameterize falloff



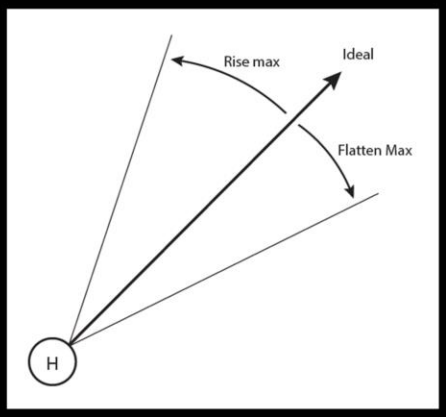
For range, we are looking for points that match our ideal line length and have the score fall off as the distance deviates from it.

[BONUS INFO]

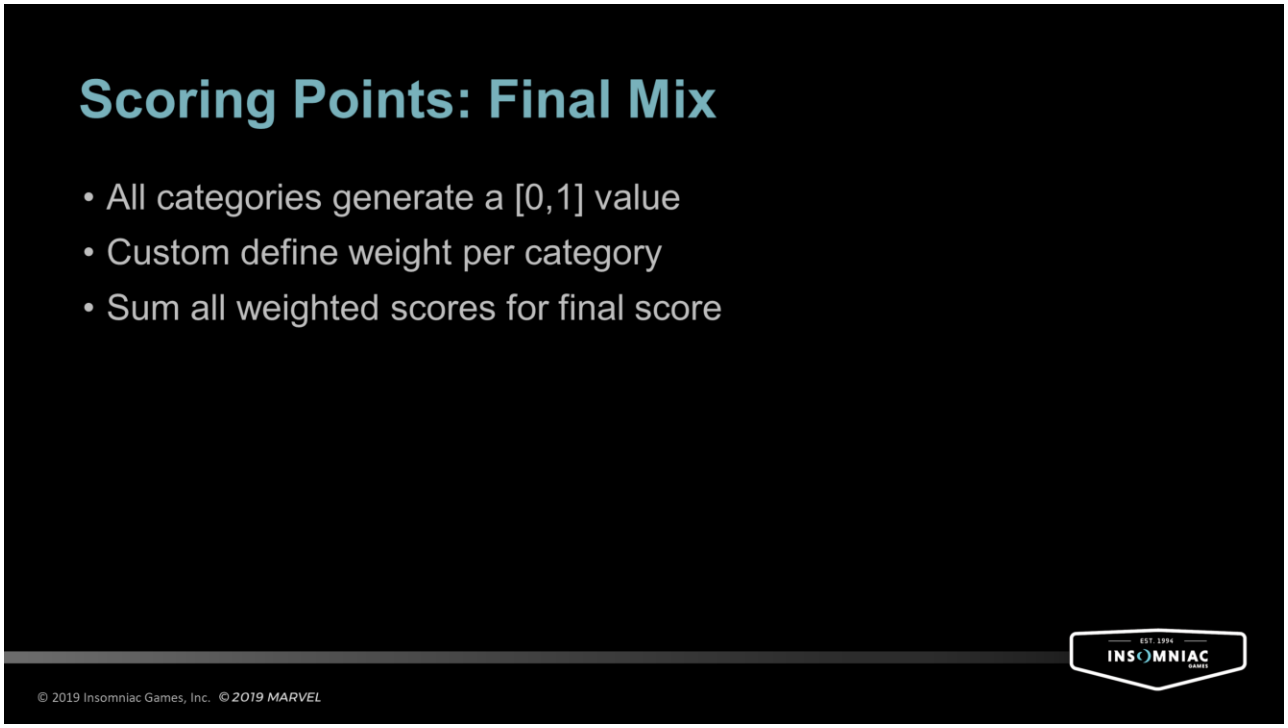
- Similar to input direction, we also adjust slightly to be more forgiving while turning
- When going straight, we prefer longer lines
- When turning, shorter lines are given a slight bump to try and let us hook corners more effectively

Scoring Points: Slope

- Scored on deviation from ideal slope
- Custom falloff for going up-vs-down



- We apply a similar falloff method for slope as the angle moves away from our ideal.



- With all of our individual scores calculated we then do a weighted sum and the highest score wins.

[BONUS INFO]

- Using a weighting scheme on the normalized element scores helped us to quickly adjust one elements influence versus another's without needing to mess with the individual elements.

- Here you can see it all coming together in-game
- [PLAY VIDEO]
- The blue boxes drawn over buildings are the markup volumes with horizontal probes representing points above us.
- You can see a little bit of debug info

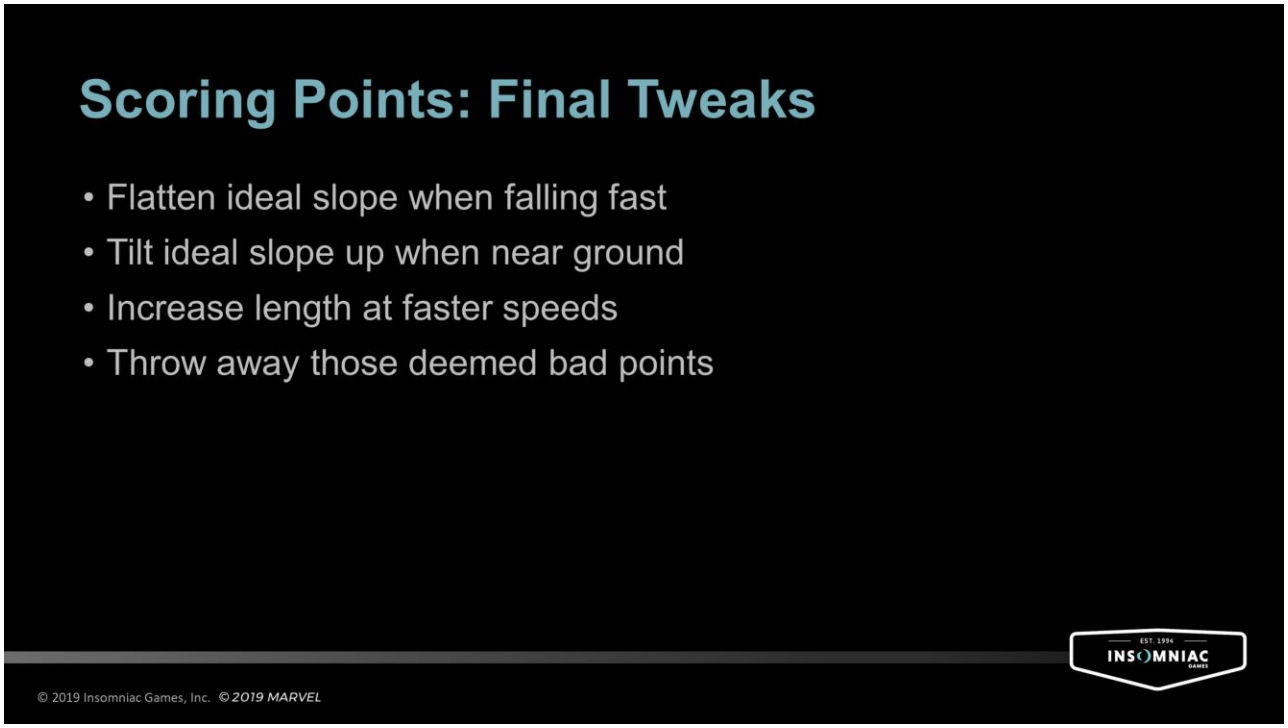
about each point’s score floating above it as well.

Swing Attach Points: Debugging

- A quick note about debugging tools, here you can see our swing history display
- This shows us not only the player path and attach points during their swings but also arrows for the input direction, camera direction, and velocity direction at the moment of

attach point selection.

- This proved incredibly useful in usability testing where we could have players swing around and just stop when a line felt weird.
    - We could then take a look at the debug history, see what the input parameters were, look at the scoring for all the surrounding points, and see if we needed to adjust the scoring algorithm.



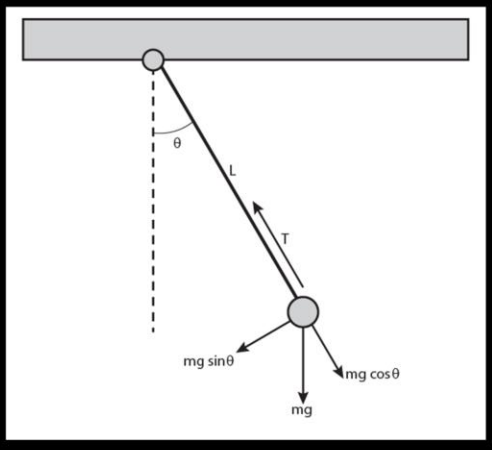
Finally, we injected a number of additional influences to help improve the overall experience as we continued development. These small items help smooth out rough edges in a variety of places while still working with the core selection process.

- [ADVANCE] We can adjust the slope based on fall direction or ground proximity to help with flow.
- [ADVANCE] We increase the line length at high speeds so that your time on the line stays more consistent.
- [ADVANCE] We throw away a number of points we deem bad because they would deviate the player away from their intended direction too much.
    - Generally speaking, we found that getting no line was better than getting a bad line as falling provided more continuity than the sudden change created by a bad swing line.

- Now that we have our attach point, we can talk about what happens when you are actually on the line.
  - As stated, we wanted to have a physics based system that we modify the results of so let's start with physics and then proceed to mess with the universe a bit.

Scoring Points: Slope

- Tension (T) = $L((mv^2)/r) + (mg*\cos\Theta)$
  - r = Length(L)
- $mg = (mg*\cos\Theta) + (mg*\sin\Theta)$

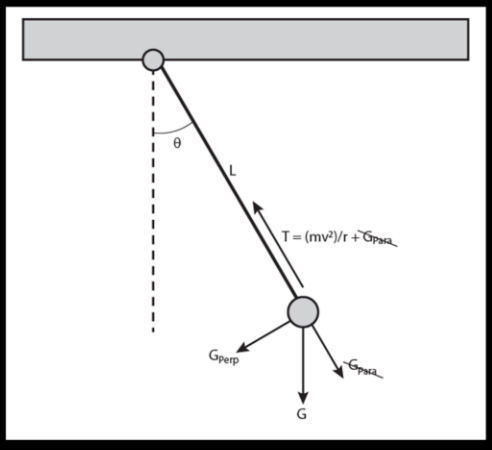


- The two basic forces involved in the pendulum are gravity and tension.
- As you can see here, tension is a factor of line length, angle, and gravity in the direction of the line
  - It is responsible for pulling the bob of the pendulum in a circle around the center of the system.

- We also break gravity up into it's two component vectors here, one parallel to the line, one perpendicular
  - The perpendicular portion of gravity represents the restoring force. This is the part that causes the pendulum to oscillate back and forth
  - The parallel portion is what's left and, in our calculations can actually cancel out part of the tension force to simplify the math a bit.

Scoring Points: Slope

- Simplify
  - *m = 1*
  - G_{Para} = Project(G, L)
  - G_{Perp} = G – G_{para}
  - Cancel opposing G_{para}
- Apply T and G_{Perp} to as forces to velocity
- 4 iterations per frame



- When actually calculating this in code, we set mass to one for simplicity as we aren't dealing with variable mass systems.
- After a bit of simplification, we apply the remaining forces of tension and perpendicular gravity to our velocity
- In order to increase the accuracy of

the simulation, we do four iterations each frame to run at a total of 120Hz

---

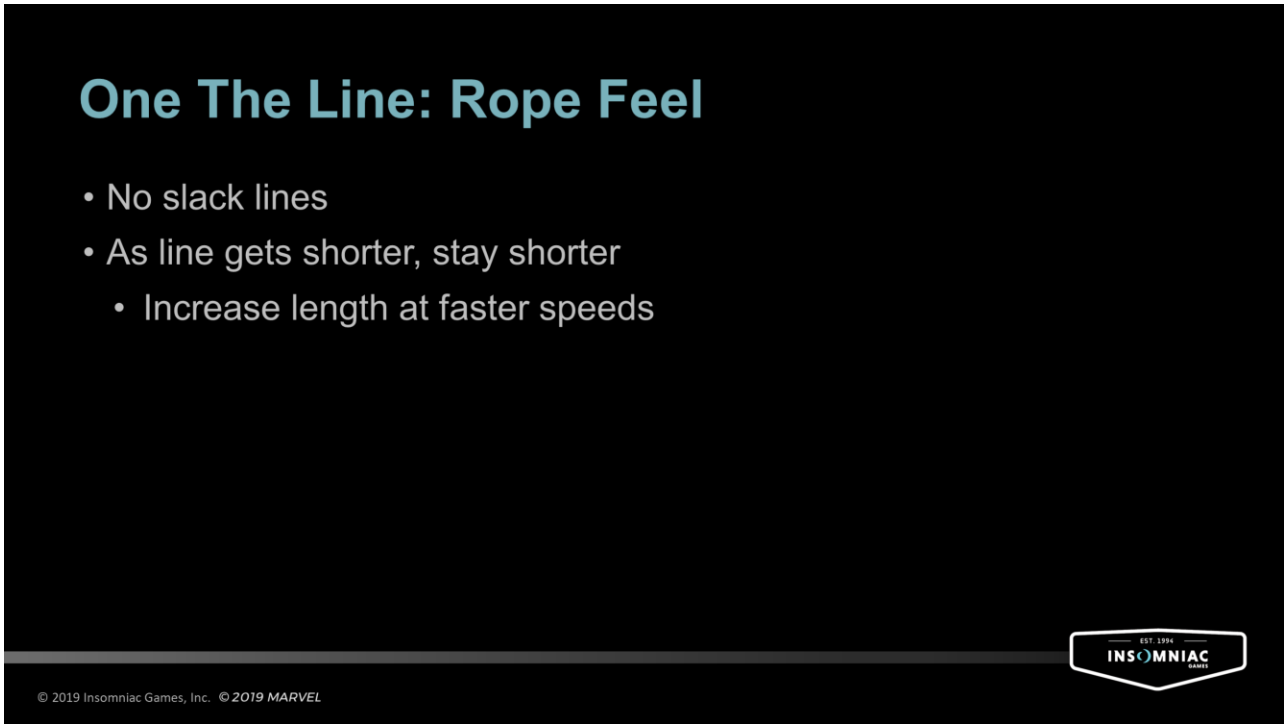
[BONUS INFO]

- In actual implementation, we apply these forces only to the portion of velocity that are tangent to the line. We then apply full gravity to the remaining velocity and recombine for a final velocity.

When we were talking about how we wanted swinging to feel the number one thing that came up was that swings should feel fluid. This lead to a couple of rules that droves a variety of other elements.

- First, we wanted to avoid slack lines
  - We wanted to keep the player feeling like they were swinging, not falling.
  - By keeping the line taut at all times it gave the player a much more predictable experience and prevented bounciness and

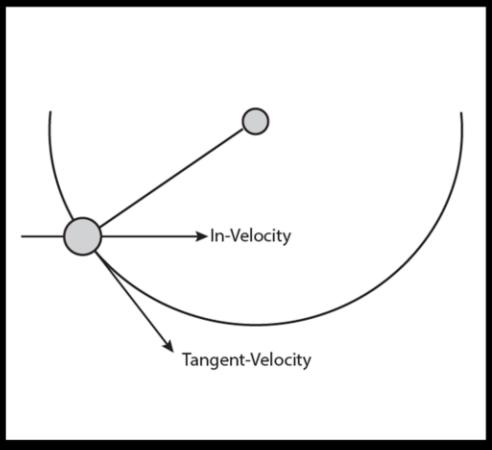
sudden changes in direction as a slack line goes taut from a free fall.



- This lead to the need that once a line is shortened, it generally stays that length.
  - We do look for opportunities to try and restore lost line length when possible but doing so too quickly feels physically wrong so it must be done with caution.

On The Line: Smoothing Out Entry

- Incoming velocity nearly always different than tangent velocity



Our first goal on getting a new line is giving the player a good experience of getting into the swing. However, the initial set up poses an immediate challenge:

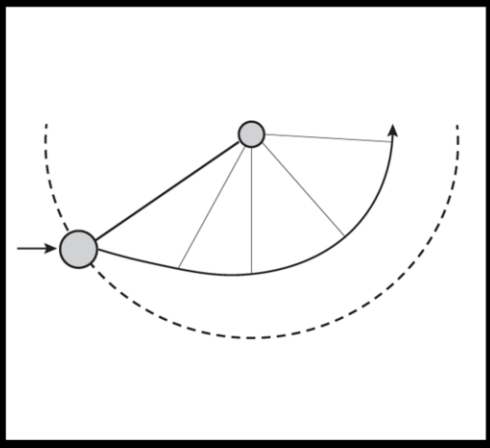
- The incoming velocity is nearly always very different than the velocity that is tangent to the swing

arc

- We don’t want to snap the velocity because that will feel like a sudden change in direction

On The Line: Smoothing Out Entry

- Working off incoming velocity directly

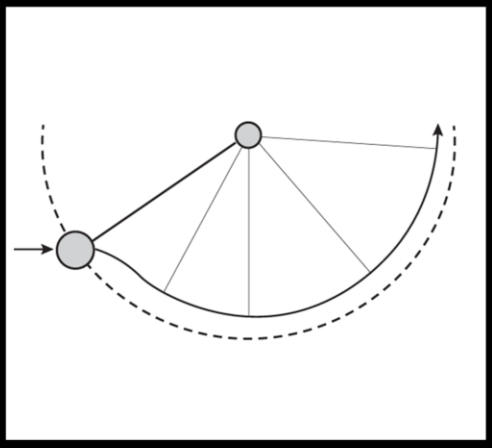


- However, if we just let the simulation play out, you get something that looks like this.
  - Because of our rule that we don't allow the line to lengthen again, the velocity running into the center of the swing arc shortens the line considerably.

- This can result in losing the swing's dip as well as causing a rapid acceleration in angular velocity that is hard for players to react to.

On The Line: Smoothing Out Entry

- Each iteration, rotationally blend velocity towards tangent



- To improve, we blend the incoming velocity towards the tangent direction of the swing arc a little bit each iteration.
- This helps maintain healthier line lengths and improves expected behavior in angular velocity.
- The player can also now feel a much

more significant drop in the swing arc compared to the previous motion path.

On The Line: Player Influence

- Player needs post-attach control
- Simple analog driven rotation speed
  - Blend in over time
  - Tune for turning with/against standard arc
  - Tune for position on arc

Once they're into the swing, the player is going to want to further influence their motion in order to move themselves around.

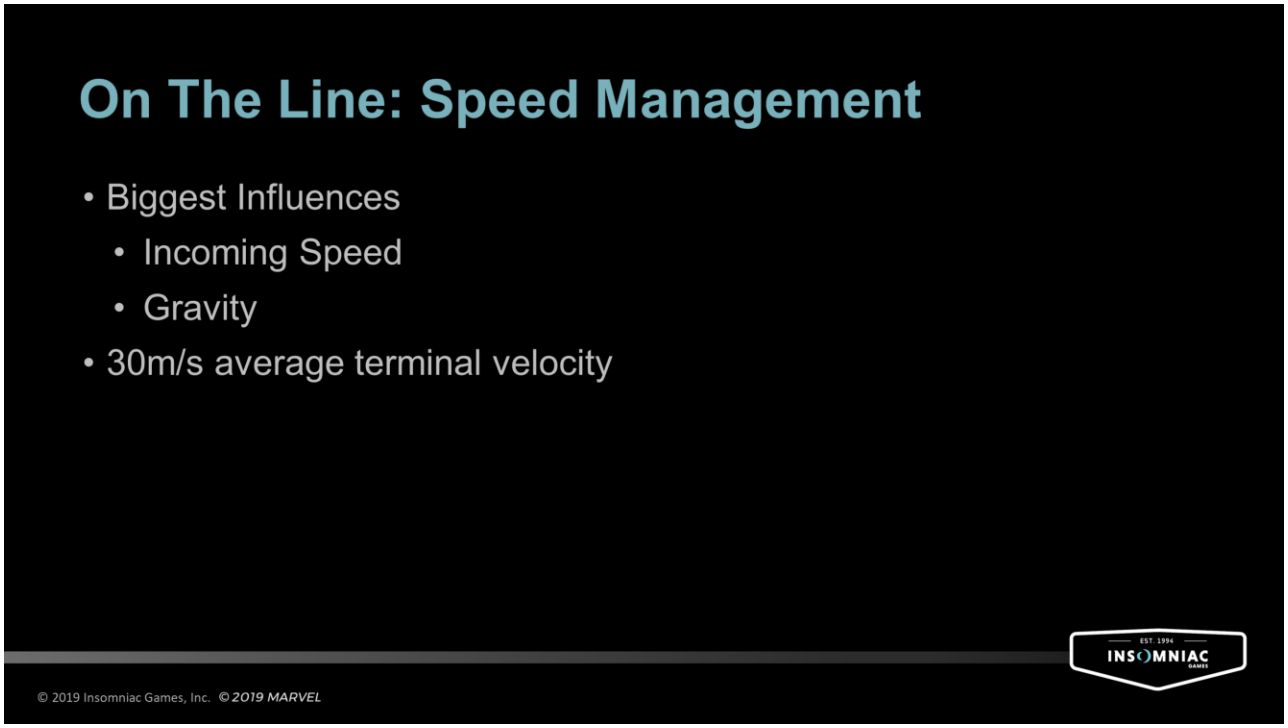
- We do this with a simple accelerated rotation of the total velocity.
- We scale the turn speed so that you have maximum control through the trough of the swing but much less

ability to turn at the swing's apex.

- We also adjust turn speed based on whether the player is turning with the natural attachment or away from it.

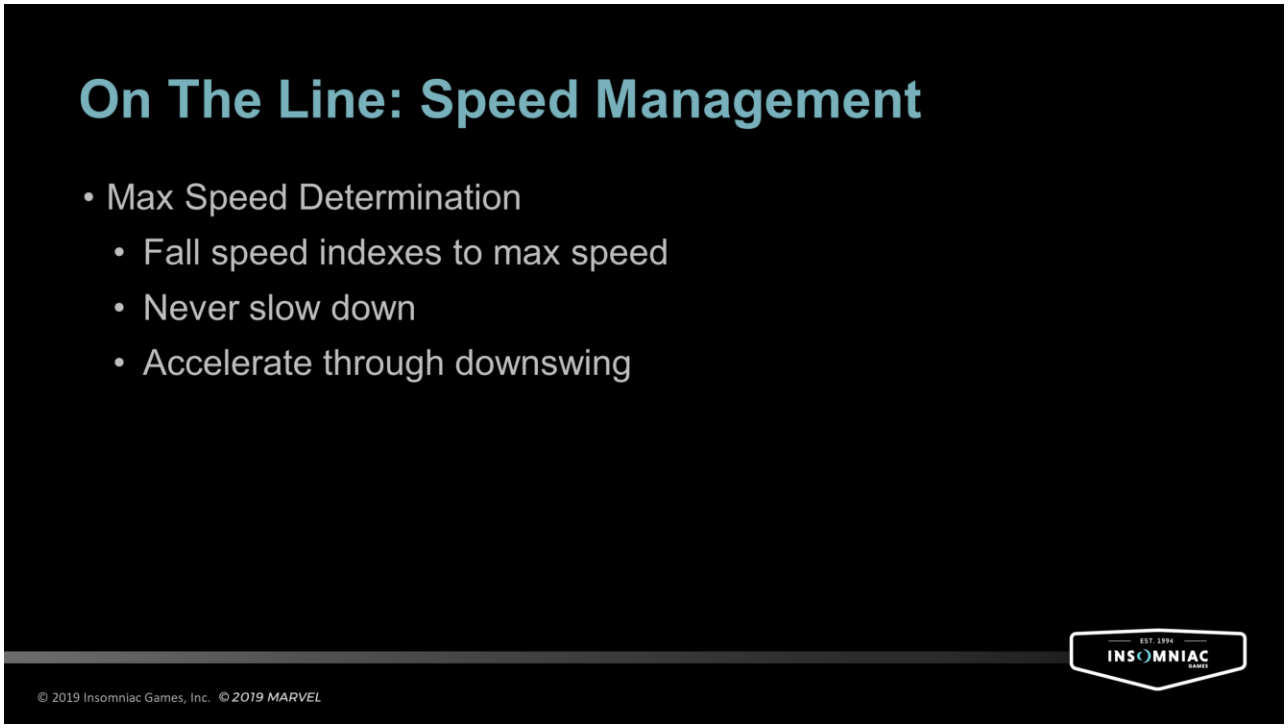
[BONUS INFO]

- We blend it in over the first few frames of the swing so that the player can straighten out the move stick if they choose and to make it feel more natural



- When it comes to managing speed while on the line, things vary greatly from swing to swing
  - The two biggest influences on the speed for any given swing will be the amount of speed the hero brings in and the gravity applied during the swing

- Due to streaming considerations we also need to stay below an average speed of 30m/s to avoid loading stalls.



- We start be deciding what our horizontal terminal velocity is for any given swing.
- As a baseline, this is done by translating fall speed into max speed while never letting it slow you down.
    - This feels pretty natural to players because they are using gravity to

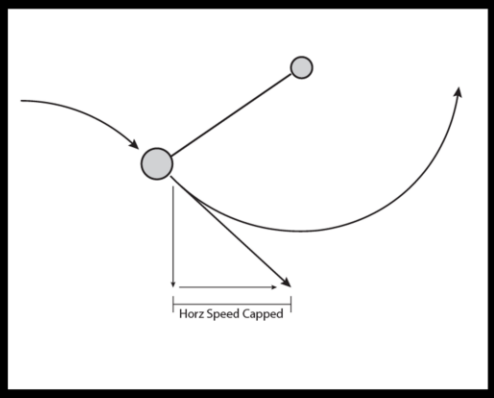
help generate speed.

- As a bonus, it helps on the streaming side because to build up that fall speed you need to be dropping for awhile which will dampen your horizontal speed.

- We then let normal swing physics accelerate you up to that max speed.

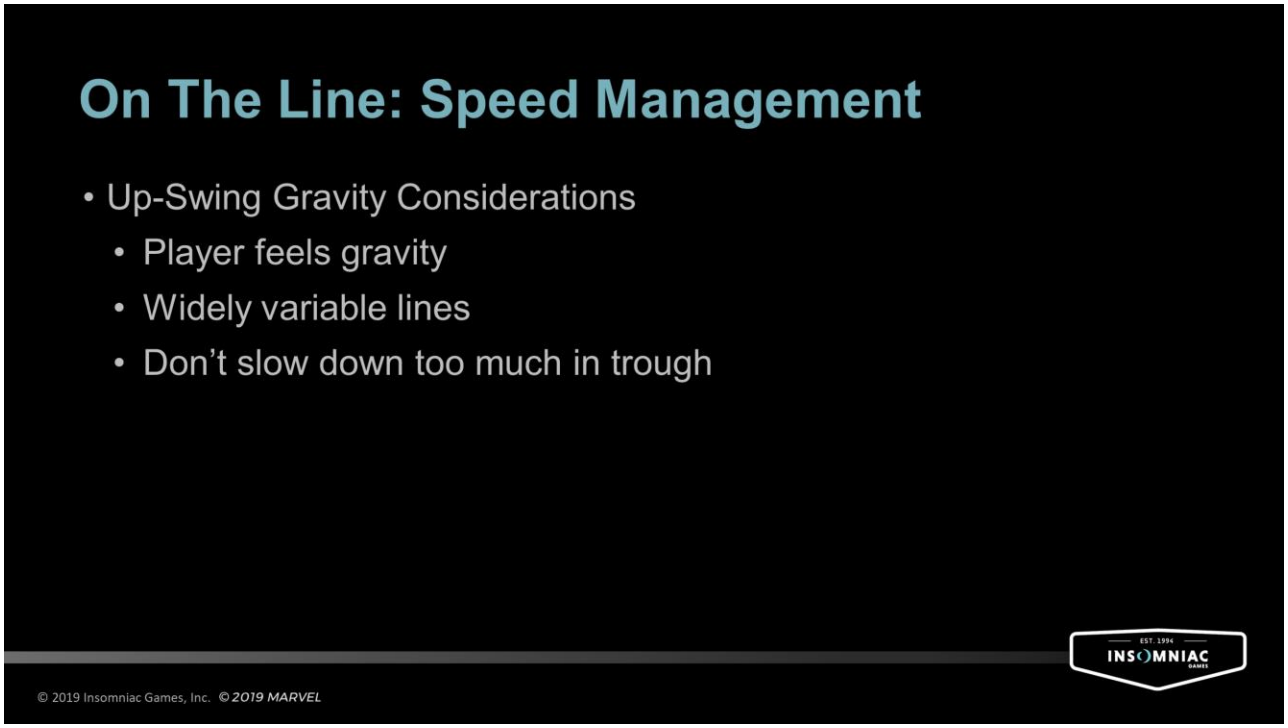
On The Line: Speed Management

- Speed Capping
  - 3D speed can exceed max
  - 2D speed cannot



- To enforce the terminal velocity, we only cap the hero's horizontal speed.
- This can have an odd side effect where you can actually slow down in 3D through the downswing but it helps you get through a long swing arc much faster
- The slowdown doesn't come into play

too often and most players are completely unaware it’s happening due to other speed cues like camera rotation, environment motion, and screen fx



On the up-swing, gravity will be slowing the hero down. However, we have a few things we want to take into account on the up-swing

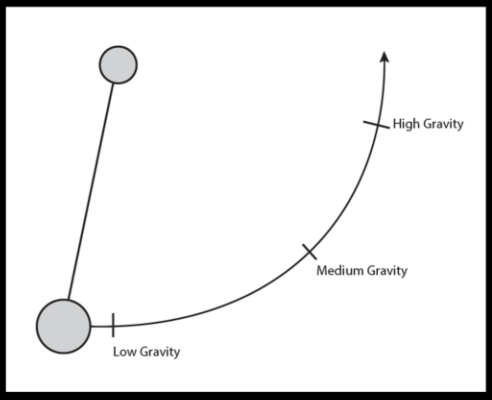
- We want players to feel like gravity is appropriately slowing them down
- However, line lengths are highly variable, which creates inconsistent

timing to the various release levels.

- We also don’t want to slow the player down too much when near the trough, instead letting them keep their hard-earned speed a bit longer.

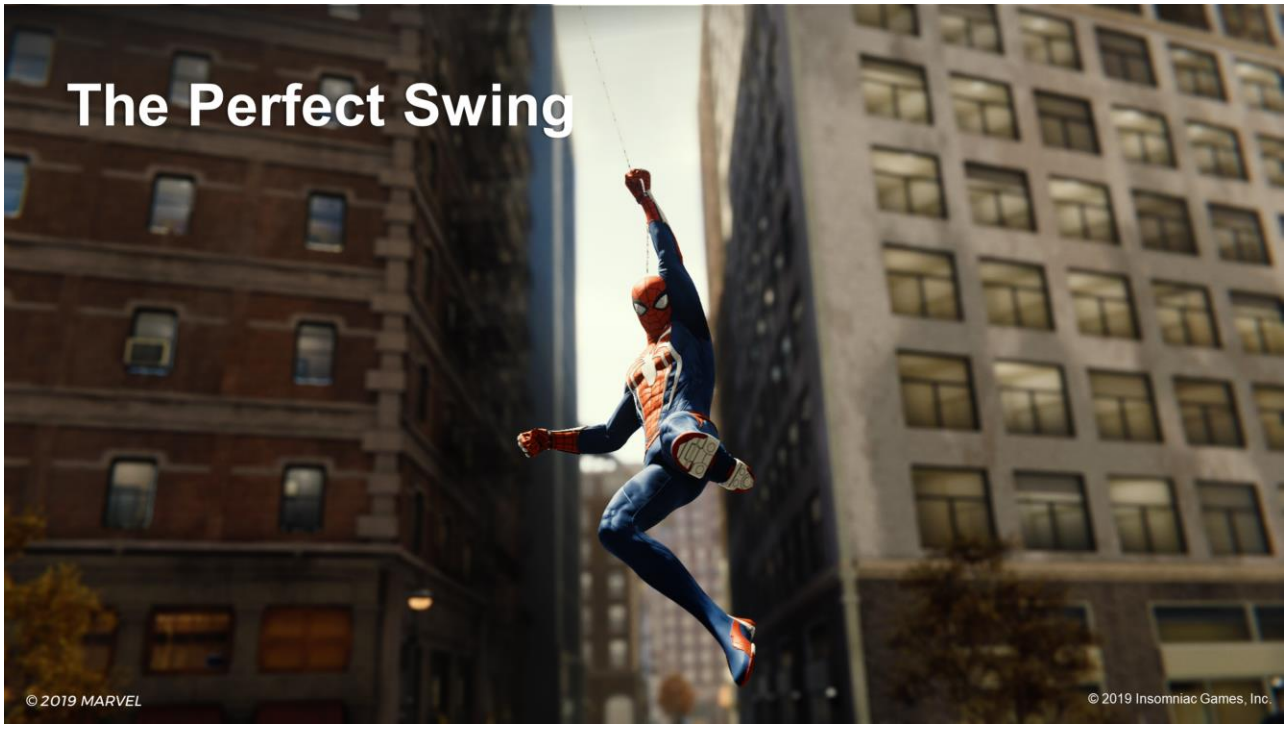
On The Line: Speed Management

- Up-Swing Gravity
  - Increase gravity with line angle
  - Increase gravity with speed



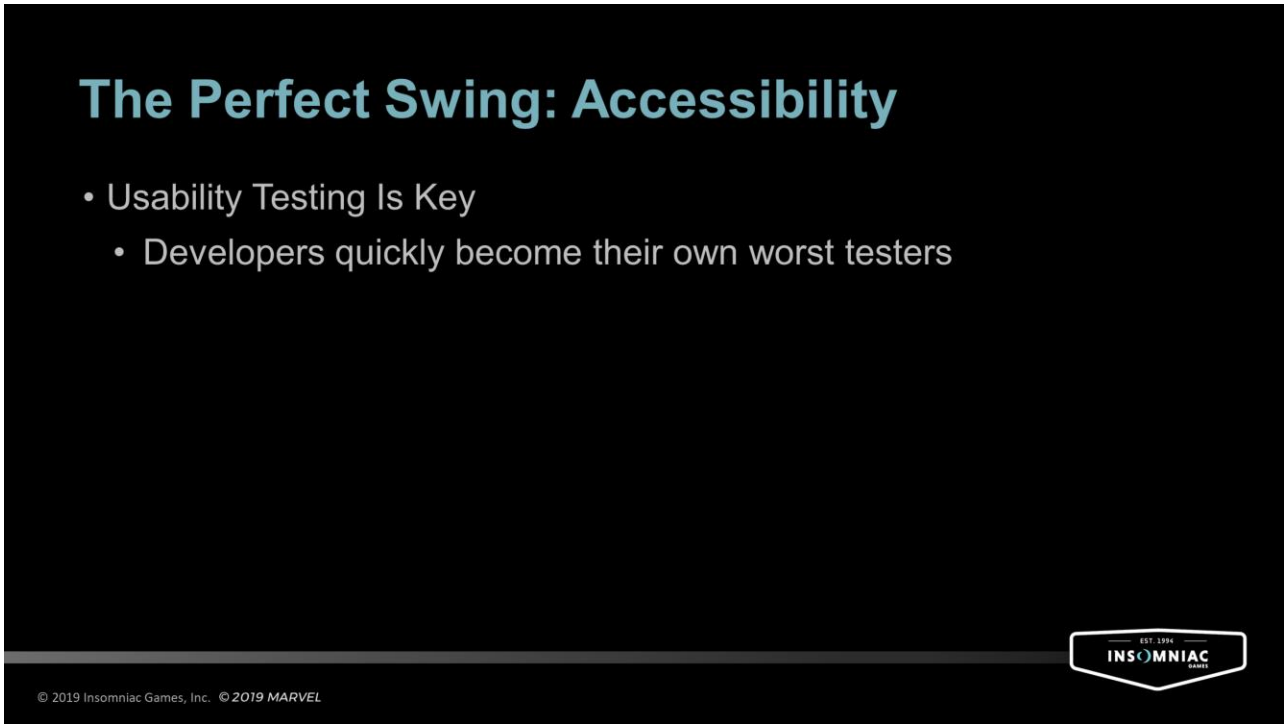
- To account for all this, we implemented a variable gravity up-swing.
  - At any given point on the up-swing, your gravity is determined by a combination of your current speed and where you are on the swing-arc.

- This helps make swing timing more consistent as well as allowing more swings to move through the full range of release points



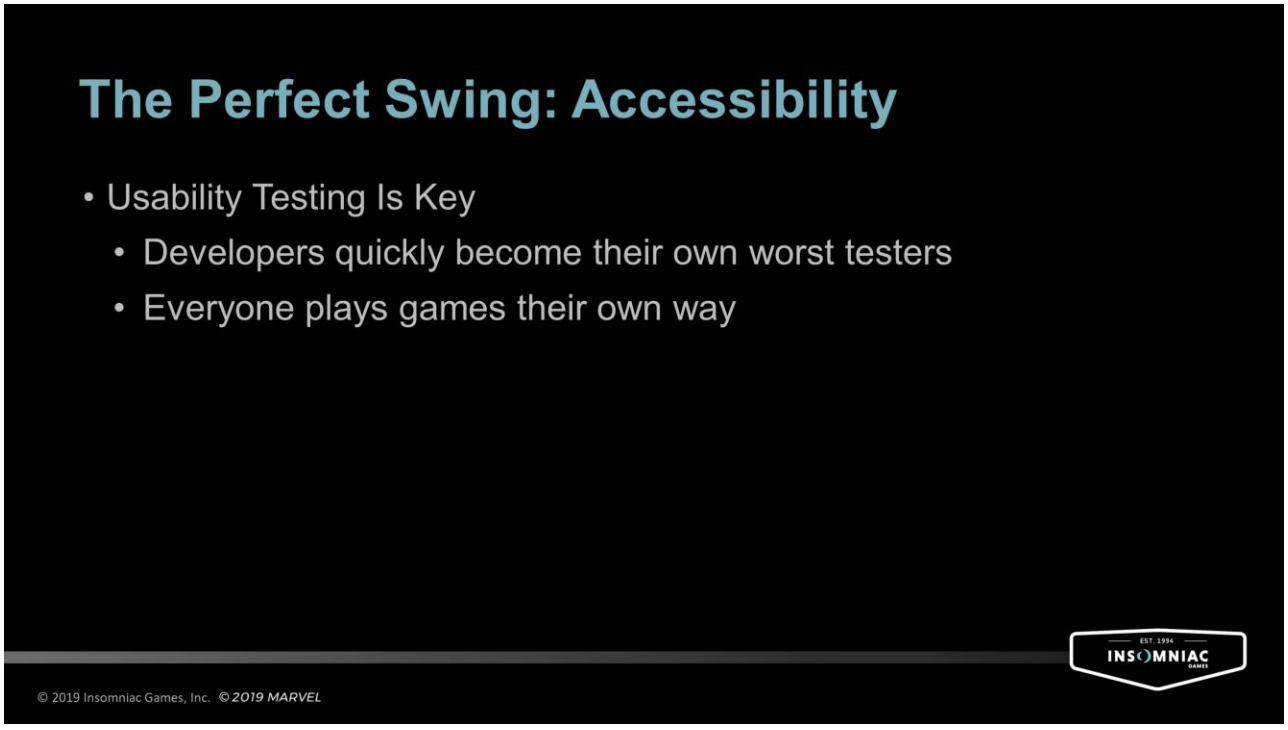
- With the basics now in place, I want to talk about our experience of refining the swinging mechanics over the course of production.
- Again we go back to our usability-driven iterative approach.
    - Per our initial goals, we wanted players to feel the momentum

and physics but also wanted swinging to be accessible.



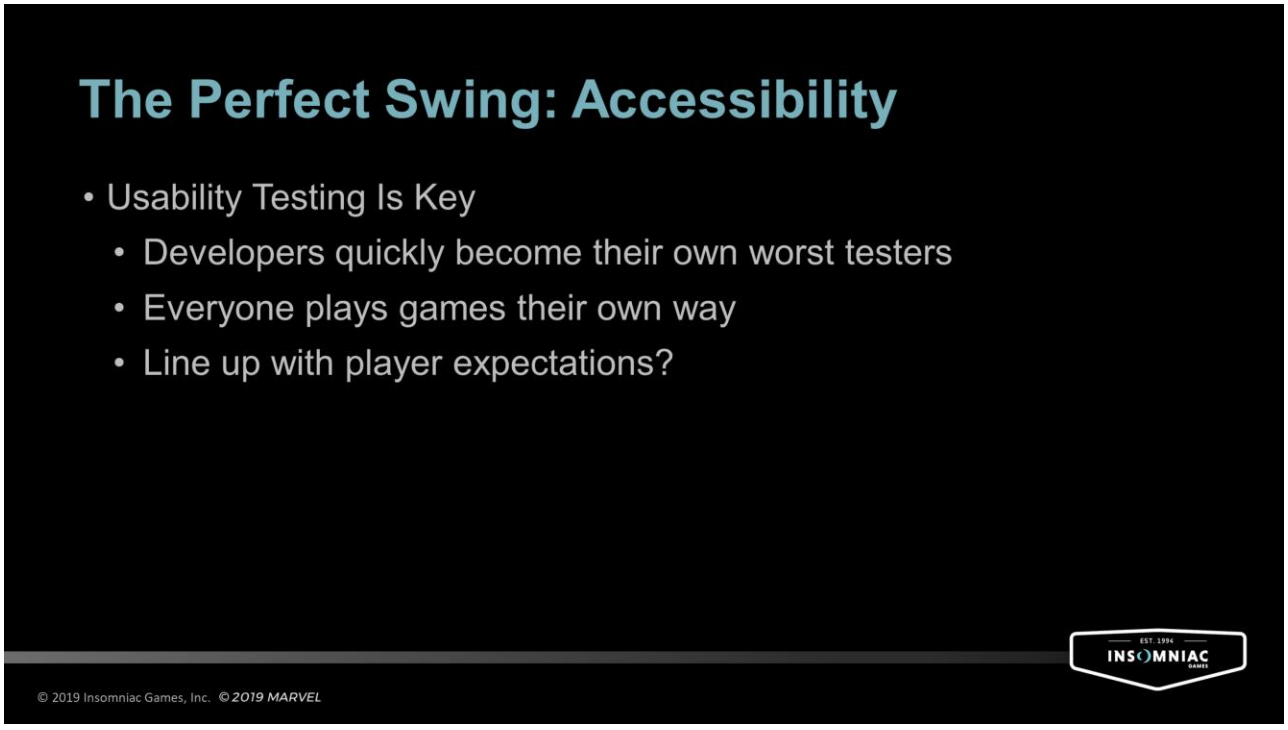
- Usability testing proves a critical element in iterating on accessibility.
- This is because developers quickly become their own worst testers
    - We know too much about how things work and are generally quick to forgive ourselves because our code is amazing and

shame on you for suggesting otherwise.



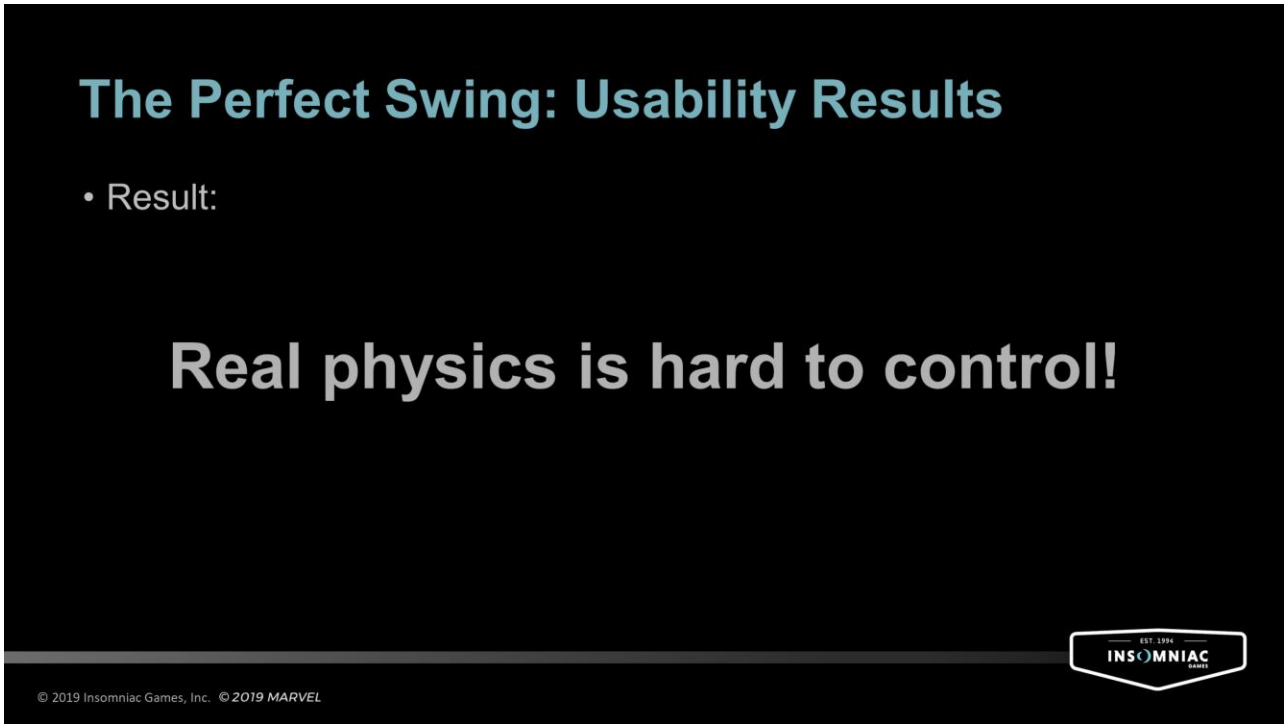
- Also, developers tend to play the game a particular way, the way it was designed.
    - Real players will interact with the game in a number of unexpected, and sometimes confounding ways, stressing elements of the system that a developer may

never encounter



- Finally, you need to make sure functionality lines up with player expectations.
  - As a developer, we may have a perfectly rational explanation for why something works the way it does, but to a player it will just seem weird, broken, confusing, or

simply not very fun.



- So, what did we find?
- [ADVANCE] Players suck at managing physics
  - Forces will often act against their desires
  - Attach points are not directly player driven leading to a lot of variability

- World geometry is complex and difficult for many players to effectively navigate while tethered to an attach point

- The end result was that players felt they lacked control over in-game results.
  - This was despite everything working “as intended” when we made the initial version of swinging.

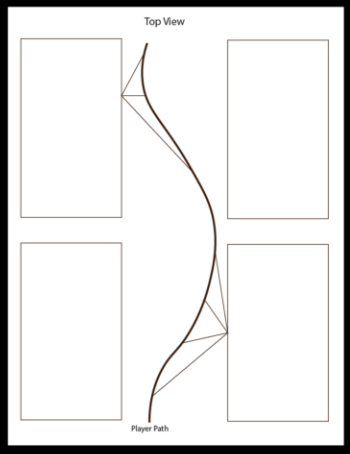
So, how did we approach solving this problem?

- We injected a number of assists to swinging to help smooth out the experience, giving them a few less things to worry about and moving the system's behavior closer to player expectations where possible.

- Our goal was to keep these as invisible to the player as possible
    - We found that if you deviated too far from real physics, players would notice and often react negatively.

On The Line: Pivot Tweaking

- UX Problem: Players felt like it was impossible to swing in a straight line.

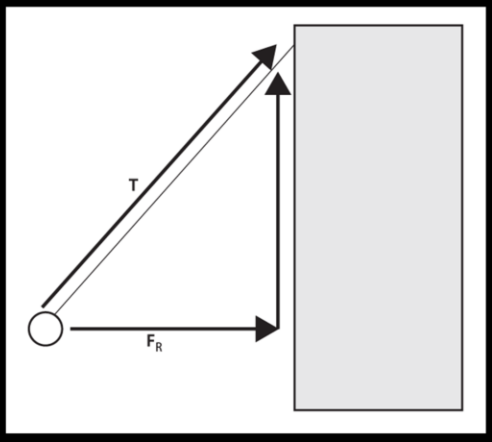


- Our first major usability problem was that players felt like it was impossible to swing in a straight line
  - The results of the simulation were creating a mismatch between player input intent and the character motion results that confused and frustrated many

players.

On The Line: Pivot Tweaking

- Reason
  - Tension and gravity forces want to settle under the attach point



- The reason this was happening is that forces were conspiring to move the character in a way that it would settle underneath the attach point.
  - The tension force has a component that moves towards the attach point with gravity working to reinforce it.

In this video you can see you what happens when the tension forces are in full effect.

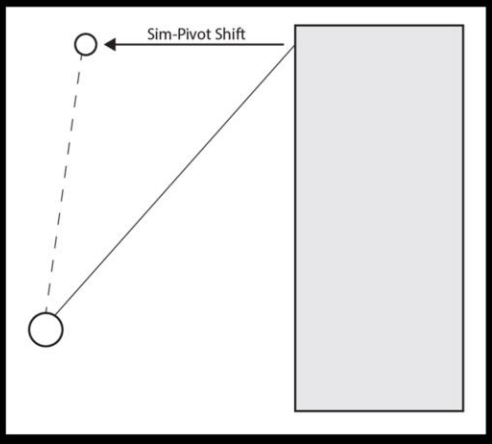
- [PLAY VIDEO]
- The player is trying to swing straight down the middle of the street but is consistently pulled sideways towards

the attach point

- When they counter-steer to get back in the middle it is likely their next attach point is on the other side and they get pulled quickly in that direction, increasing frustration
- To top it all off, this zig-zagging motion will also slow down a player's straight-line speed

The Perfect Swing: Pivot Tweaking

- Solution: Shift the simulation pivot
- Blend back due to time or turning.



- So, how do you fight gravity? Cheat.
- We move the simulation pivot away from the wall into a position that will help us swing straighter while leaving the visible attach point on the wall.
- We then blend the pivot back into position over time or when the player is turning.

    - This helps ensure that turns pivot on the correct position and that if you settle you settle under the visible attach point.
- This motion was detectable to players but we found that most felt it actually worked more as they expected, aligning input desire to results.

In this video, pivot tweaking is fully active.

- [PLAY VIDEO]
- The player can maintain a more consistent center line and build speed more easily.
- Directional intent is transferred more

cleanly to motion allowing the player to focus on other aspects of traversal as desired

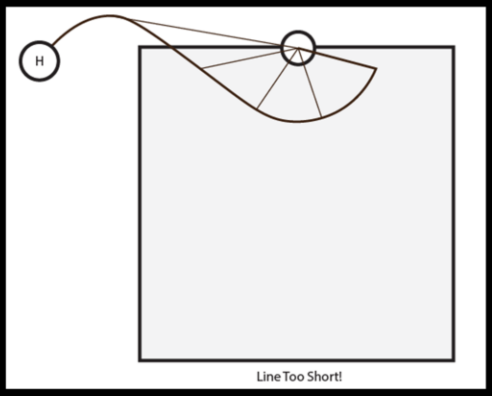
The Perfect Swing: Pivot Tweaking

- UX Problem: Players feel they can't get a swing line when their body is at or close to building top height

- The next usability problem was that players felt they were not able to get a swing line when their body is at or near the top of a building.

The Perfect Swing: Pivot Tweaking

- Reasons:
  - Attach point required to be a minimum height above you



- This was happening because we required an attach point to be a minimum height above you.
  - We do this because it prevents you from flying over the pivot or requiring a velocity change that would feel unnatural and sudden.
  - However, this rule was

completely hidden from the player and there was no reason for them to expect or anticipate it.

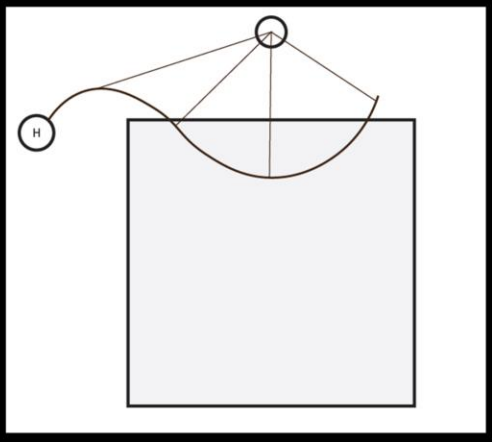
The Perfect Swing: Pivot Tweaking

- More Reasons:
  - Poor in-air spatial awareness
  - Players are impatient

- Some additional reasons this can occur is that player perception of the character position versus the world can be pretty murky when in the air.
- Also, players are just impatient when falling without being able to attach to something.

The Perfect Swing: Pivot Tweaking

- Solution: Allow pivot to rise beyond markup
- Bonus: Improve existing lines as well



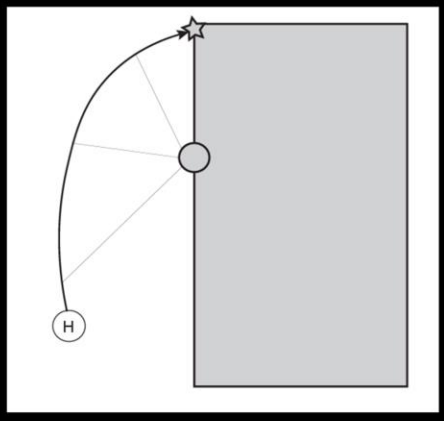
- So what did we do? Well, once a cheater, always a cheater…
  - We secretly allow the simulation attach point to rise above the building edge
  - This would raise the valid attach point ceiling and let people attach sooner.

- It also had the nice side effect of improving lines that were previously valid but not great.

- Player awareness of this move was nearly zero. It's actually pretty hard to notice a vertical shift in the pivot point when swinging under it.

The Perfect Swing: Steering Assist

- Usability Problem:
  - Players would often clip edges of geometry



- Our next usability problem is that players were often oversteering when trying to turn and clipping the edges of buildings.

The Perfect Swing: Steering Assist

- Solution: Adjust to world
  - Slide around edges
  - Detect edges and side-gaps
  - Dampen turn speed to avoid over-steering before gaps

Our solution here is to do a number of collision tests to scan the world for problem cases.

- We send a cone of checks forward to look for things we might run into so that we can nudge the player slightly left or right around them
- We also send a series of checks

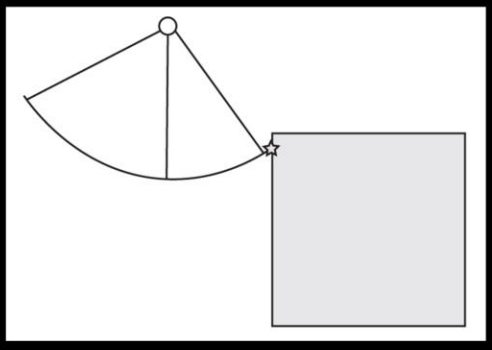
sideways looking for upcoming gaps in surrounding walls.

  - This gives us a clue that there we might want to dampen steering values so that players don't oversteer in anticipation of making a turn and slam themselves into the wall instead.

- These two assists combine to give players a hugely empowering feeling of "just made it!" without really noticing how we were helping them.

The Perfect Swing: Line Shortening

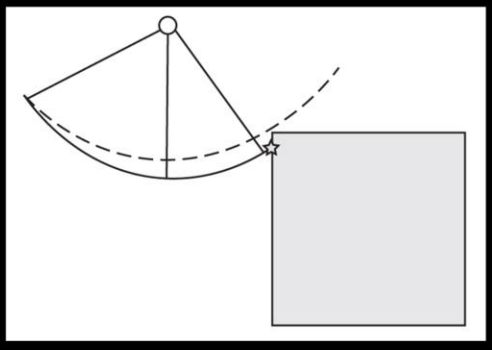
- Usability Problem:
  - Building top clipping



- Yet another consistent problem was that players would often have a line that was just long enough to smack them into the top off a building they were hoping to clear.
  - This was particularly bad because players had no control over their line length so they were quick to blame us for their troubles.

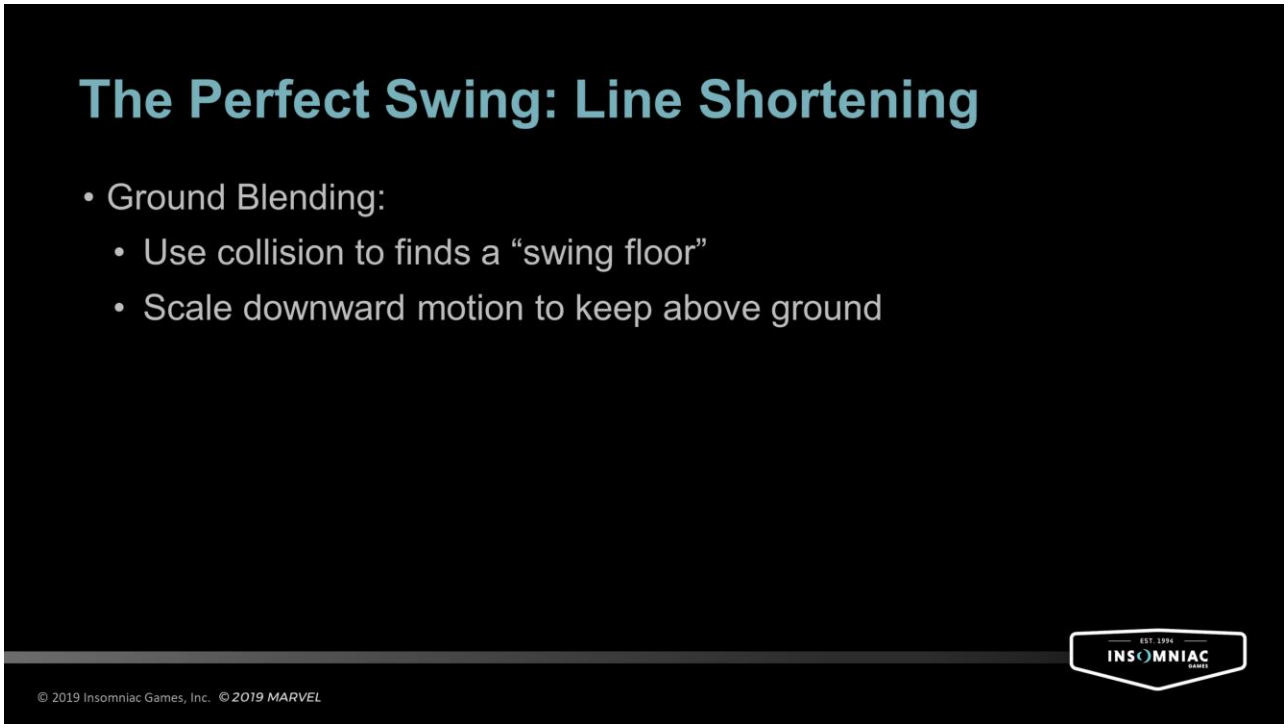
The Perfect Swing: Line Shortening

- Solution: Shorten the line



- The solution here is to shorten the line dynamically in order to get it just short enough to clear the building.
  - We would use collision to scan ahead for the maximum upcoming height and then slowly raise the player until the new line length was short enough to clear

the building.



- One final element to talk about is how we shorten the line to keep players above the ground.
  - We do this because both our ground and rooftops are so littered with cars and props that trying to navigate a swing through them would be impossible at

speed.
    - We also didn’t want players face-planting into the ground all the time, instead getting that street-sweeping swing that can feel so good.
- Similar to clearing buildings, we would use collision to get the ground height and then scale vertical motion to keep you above the ground.
- Players were pretty aware this was happening but the vast majority preferred continued motion over slamming into windshieds.

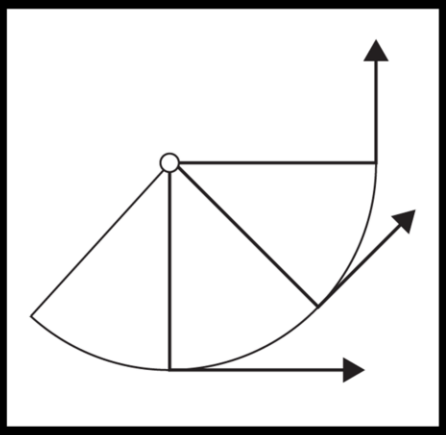
- Once a player's time on the line is complete it's time for what ends up being one of the most satisfying elements of swinging: Flinging yourself off the line and across the world.
- When doing so, we have two sets of data for letting go of a line: releasing

and jumping

- Jumping allows players to generate more height and speed while getting more acrobatic animations
- Releasing has simpler animations and allows the player to take a follow-up action more quickly

Swing Exit: Pendulum Release

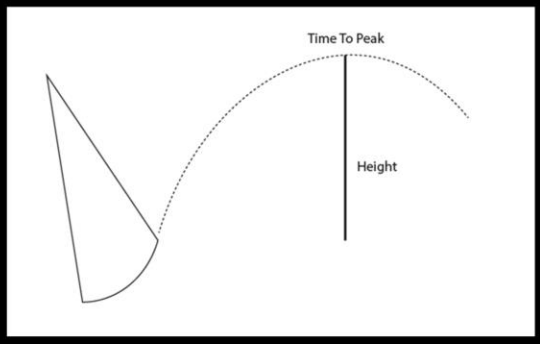
- Pendulum has analog exit points
- Release low for more horizontal speed
- Release high for more vertical speed



- One of the really nice gameplay features of swinging is the inherent analog nature of how a release point translates to an exit direction.
  - A player can choose to release at a low point to get more speed or hold on a bit longer for more height.

Swing Exit: Metric Limited

- Version 1.0: Metric Driven Approach
  - Define fixed metrics for each release point



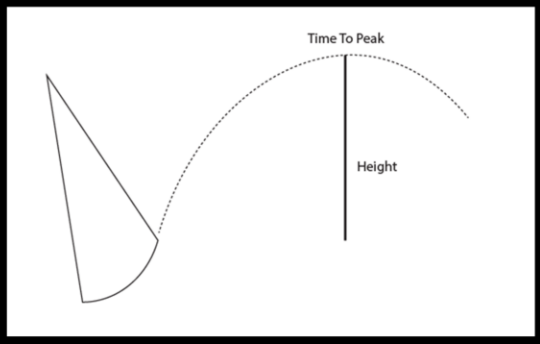
- Our first approach to swing releases was very metric driven, something that seemed sane based on Insomniac's platforming history.
  - We would define the desired jump height and time-to-peak for various release points and speeds along the swing arc and blend

those values for any release point.

- The goal was to make jumps feel predictable and consistent.

Swing Exit: Metric Limited

- Result:
  - Jumps felt too heavy
  - Player feedback of artificial ceilings and drag
  - Not enough reward for high speed releases



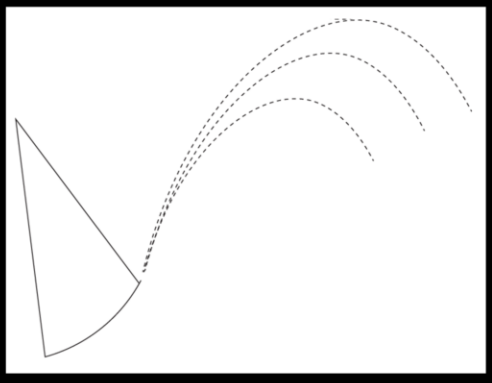
- We had this model for nearly two years and were getting mostly positive feedback but there was always an undercurrent pointing at a lack of player satisfaction.
  - Players were feeling like jumps were too “heavy” and that their momentum on the swing line was

not translating to their jumps.

- This was because the metric defined approach was changing gravity significantly for any given release point based on player speed to hit that jump-height target.
- Instead of feeling rewarded with an expected amount of “fling” players physics-feel was being disrupted and they felt like they were being penalized somehow

Swing Exit: Free Release

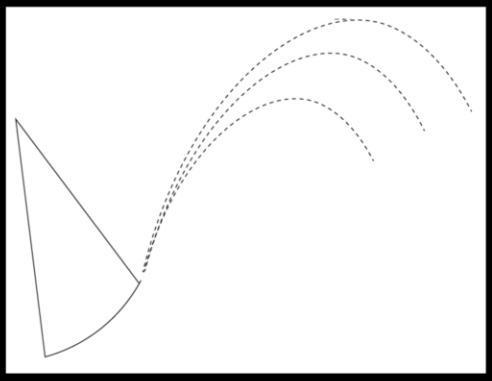
- Version 2.0: Physics driven approach
  - Abandoned arc-defining metrics
  - Fixed gravity for each release point
  - No additional limitations on jump height, distance



- To fix this, we moved to a more free-release model that let go of the controlling metrics in order to keep a fixed gravity for each release point.
- This was the only design-driven factor into the jump simulation as we wanted to remove anything else that felt artificial

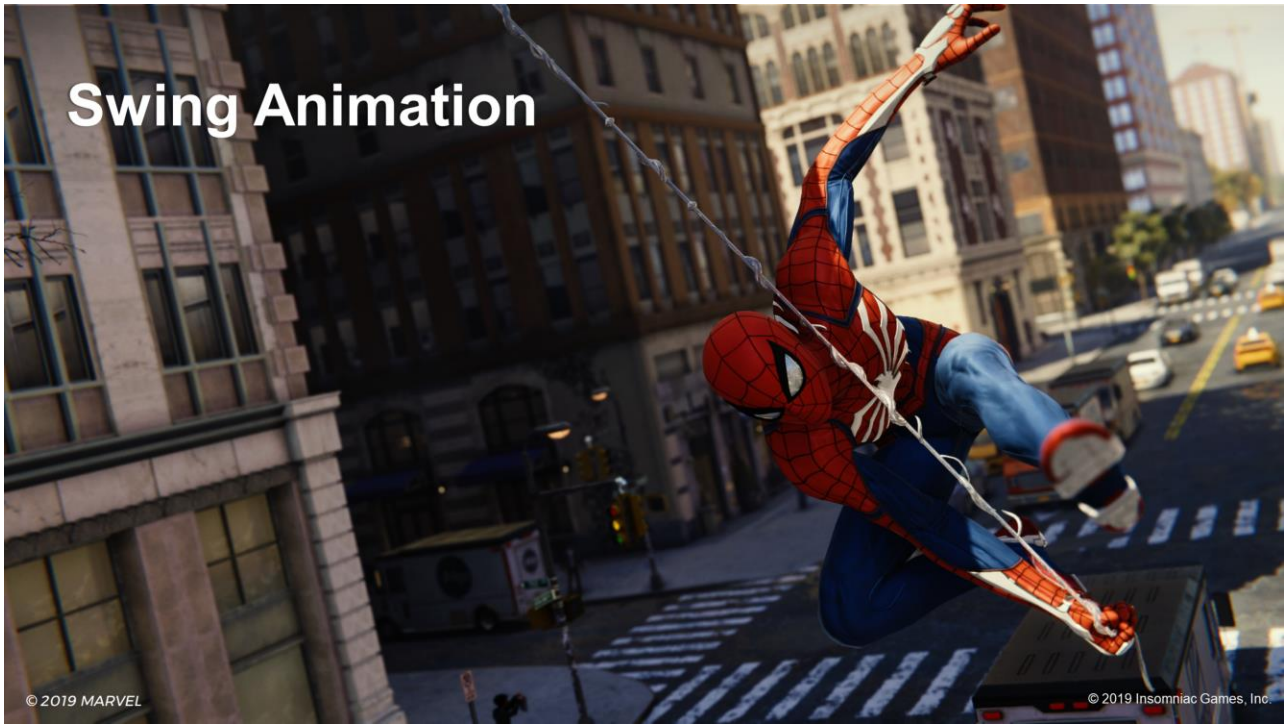
Swing Exit: Free Release

- Result:
  - Translation of swing speed to jump was much more tangible
  - Immediately increased fun factor as players were able to fling themselves around.

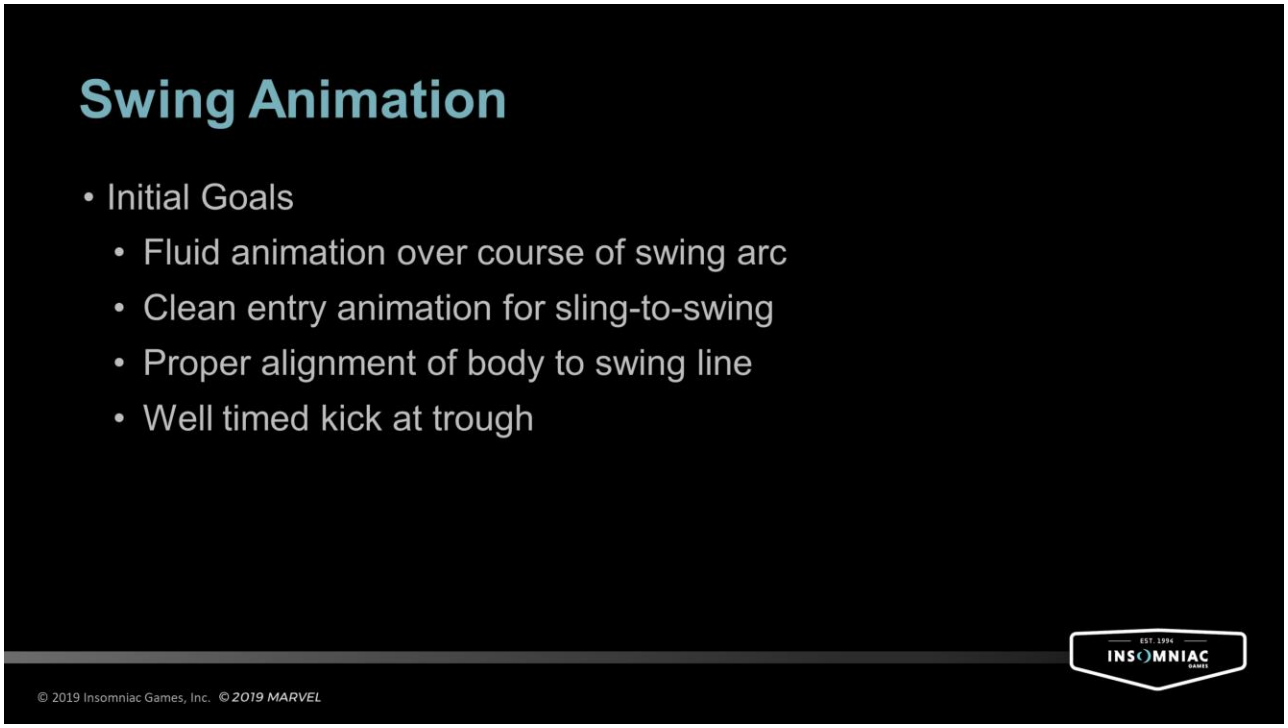


- Once implemented, the change created a much more natural translation of swing speed to jump size that better matched player expectations.
- We saw an immediate increase in fun factor as players were now able to fling themselves across the world

with abandon which lead to a greatly increased sense of speed and joy.



Moving on from the simulation, I want to talk a bit about how the animation for swinging is implemented on the code side. This will cover the mechanics of how we pick animations to play and adjust them to more accurately represent the underlying simulation.



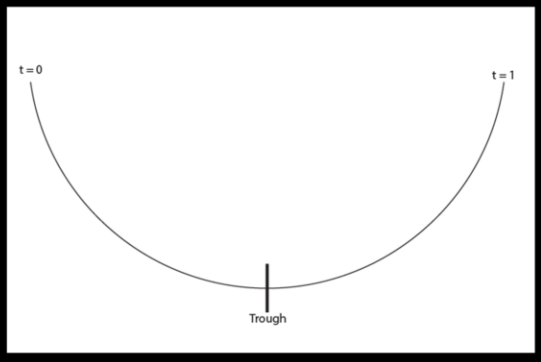
- When we first started talking about swinging the gameplay and animation groups laid out some high level goals for how we wanted the animation to look and feel. This included:
  - [ADVANCE] Having fluid animation over the course of the

swinging arc.

- [ADVANCE] A clean entry animation from our in-air pose through the sling and into the swing
- [ADVANCE] The character’s body needed to properly align to the web line.
- [ADVANCE] We needed a well-timed kick at the trough to sell momentum and as a player cue for where they were in the swing arc.

Swing Animation: Version 1.0

- Fluid animation through arc
  - Swing arc is single anim
  - Kick timed at trough
  - Code drives animation time



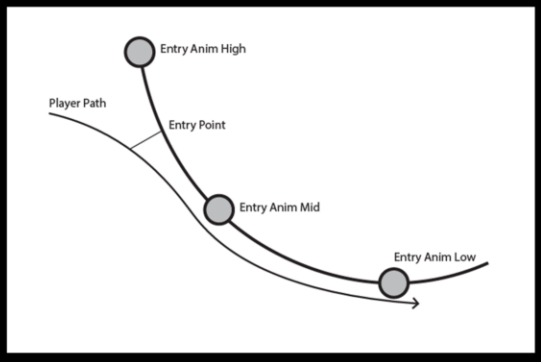
- In our first iteration of animating the swing we approached the seamless animation through the swing arc by actually using a single animation for the entire swing
  - The animator created an on-the-line animation that started at the drop-horizon and went all the

way to the apex horizon

- In code we would then drive the animation time based on where the player was in the swing arc on a given frame

Swing Animation: Version 1.0

- Clean Entry
  - Create intro-animations at three points in swing
  - Dynamically blend animations to match character position

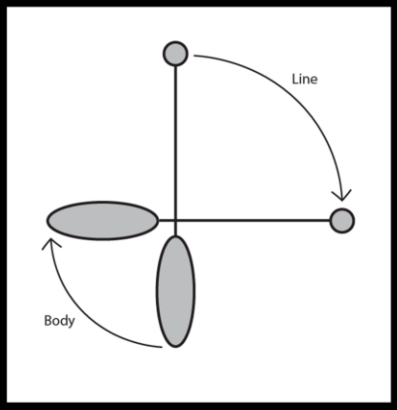


- In order to get our clean entry animations we created separate swing-intro animations for different points along the swing arc
  - These animations would end pose-matched with the full-swing animation so that the blend to the swing would be seamless

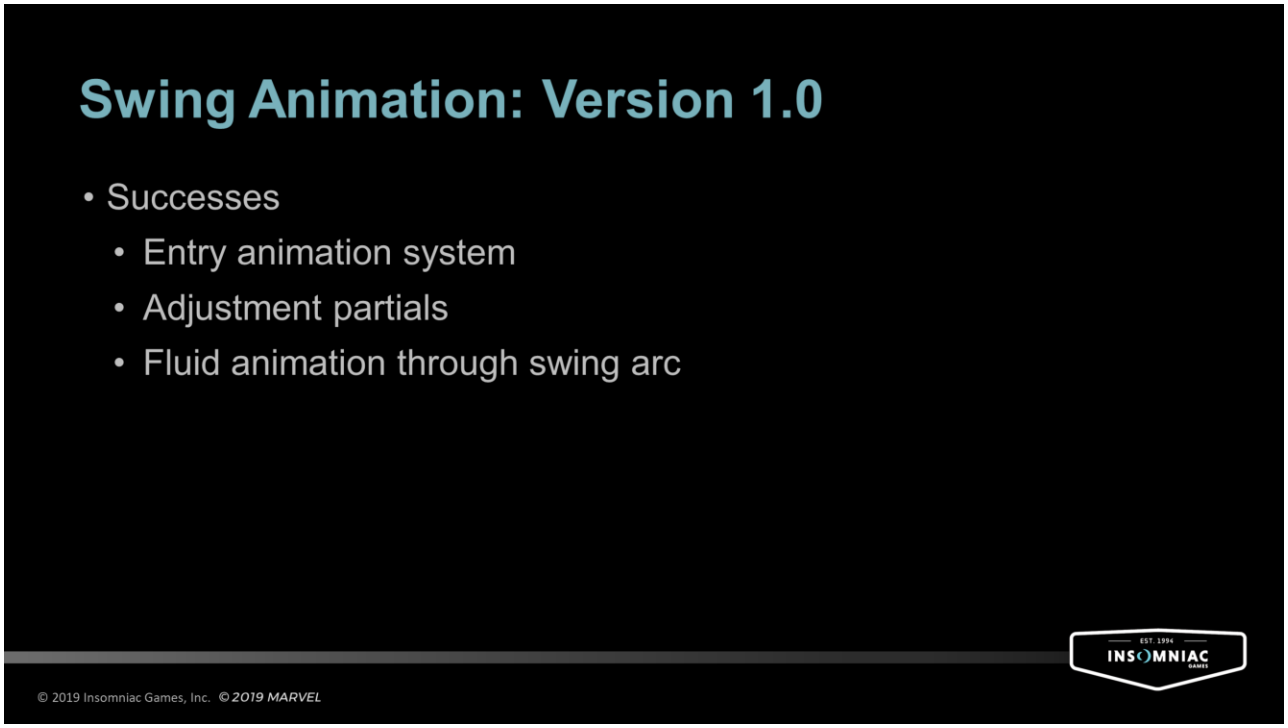
- When a player began a swing, we would calculate where they were in the arc and blend between those different animations to get an intro for that angle
- We would then continue to dynamically update that blend so that the final pose was a match for the swing animation we would play after

Swing Animation: Version 1.0

- Body Alignment
  - Partial animation rotates body
  - Dynamically apply
  - Similar process to match arms

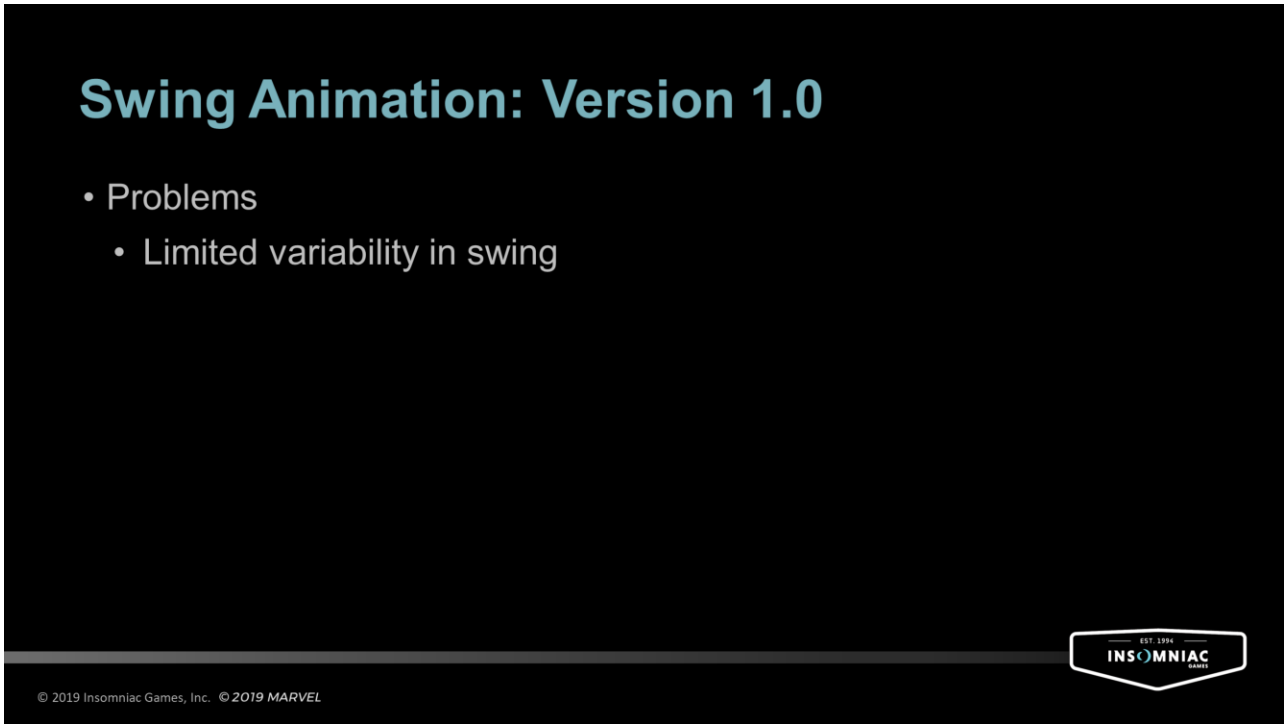


- In order to get the body to align without having to make duplicate swing-arc animations, we deployed partial animations to adjust the final body angle
  - We would apply a similar process to adjust the character's arms for final fixups



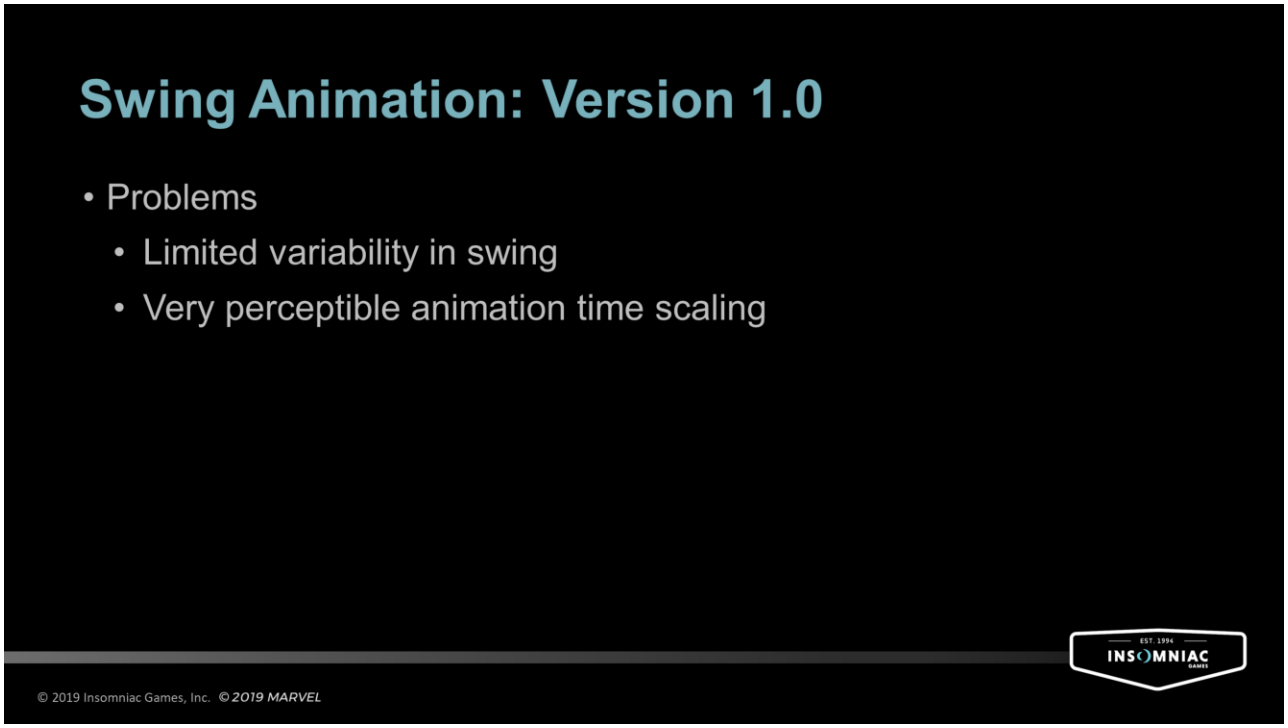
- After iterating with this model for most of production, we had some things that worked just as expected. These include:
  - The entry animation system worked well, allowing smooth entry and timing into the swing animation

- Our adjustment partials did a good job at aligning the character to the swing line with minimal additional animation required
- And finally, the animation through the swing arc was fluid.



- However, the system presented a few limitations that were capping the animator's creativity
  - First, we had limited variability in the a swing.
    - Getting any variation in this model required animating an entirely new swing-arc from start

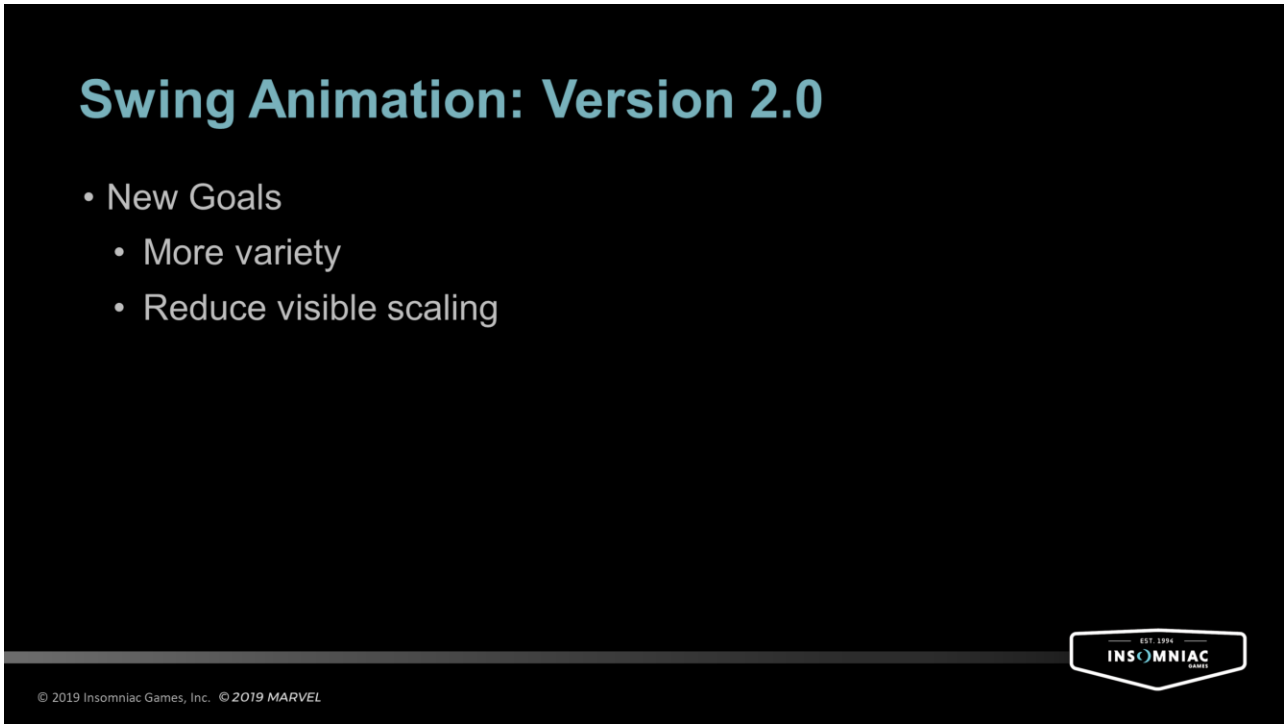
to finish



- Next, we had very perceptible animation time scaling
  - Line lengths and players speeds are highly variable which was born out in highly variable arc-speeds.
  - With a fixed number of animation frames scaling over a

variable time range we would get easy to feel time scaling.

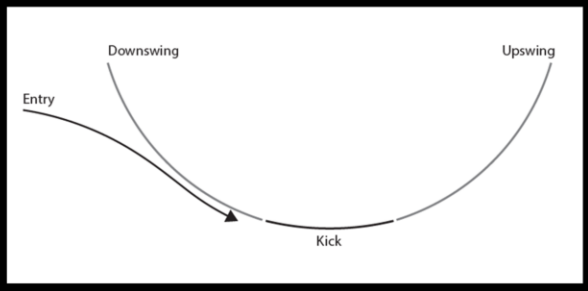
- This was especially evident when we would get slow kick throughs that lacked impact on longer lines and slower swings.



- So now we had new goals, enable variety and reduce time scaling.

Swing Animation: Version 2.0

- Break swing into separate pieces
- Continue scaling downswing and upswing
- Play kick based on dynamic timing

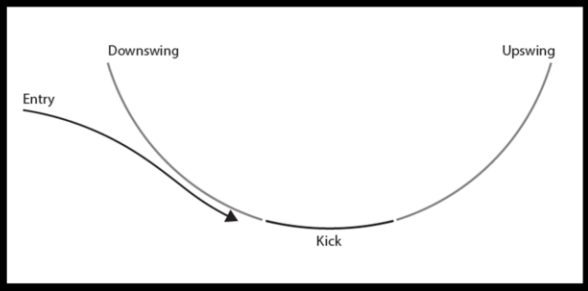


- We did this by breaking the swing arc up into pieces that we could animate separately.
  - We would animate the down-swing and up-swing separately all the way to the trough letting us overlap the kick whenever we wanted.

- We continued to drive the animation time as before on the up-and-down swing animations as they weren’t as susceptible to visible scaling.

Swing Animation: Version 2.0

- Break swing into separate pieces
- Continue scaling downswing and upswing
- Play kick based on dynamic timing



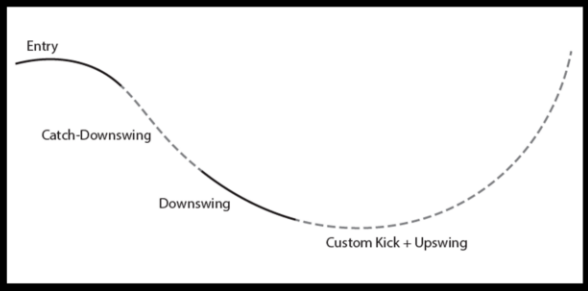
- Once in a swing, we would estimate the time-to-trough during the down-swing and trigger the kick animation at just the right time in order to be able to play it at full speed. This instantly made our kicks feel more powerful and consistent.
- It also meant that we could plug in

a variety of animations into the kick portion without needing to re-animate the rest of the swing.

- But we weren’t done yet, our animators still wanted more.

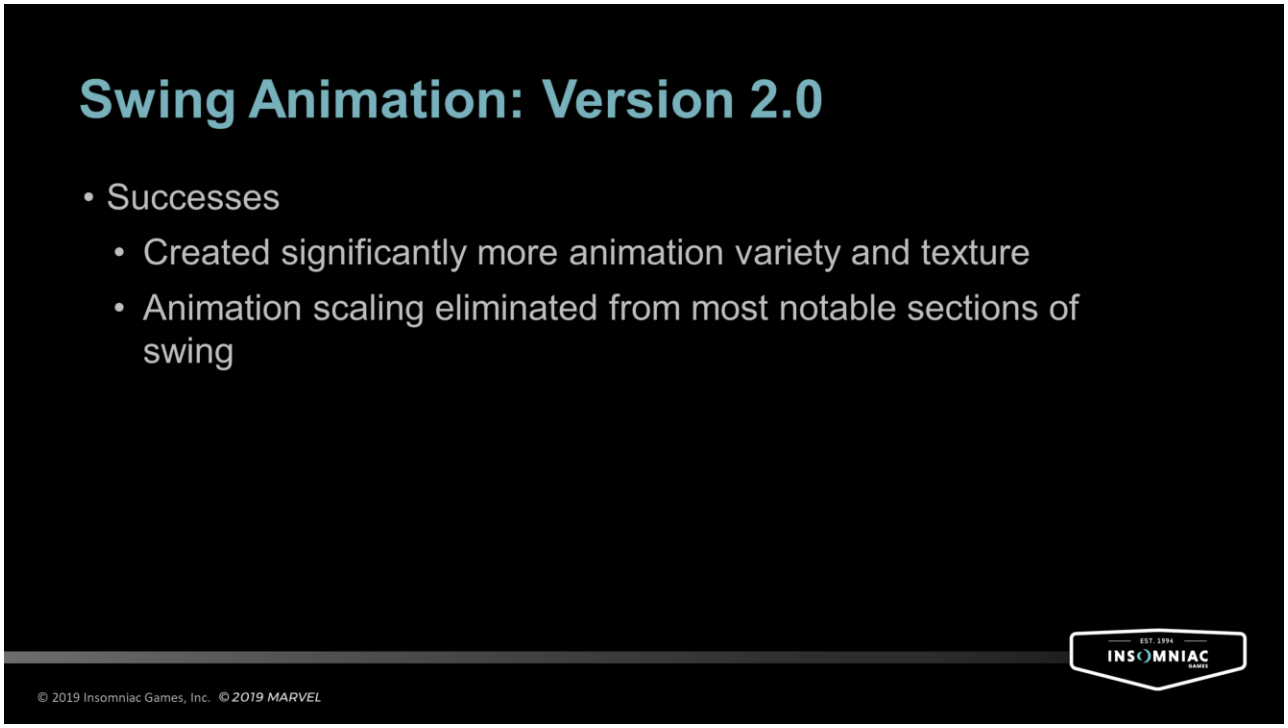
Swing Animation: Version 2.0

- Additional construction variety
- Custom kicks with built in upswings
- Hard “catch” variations of down swing

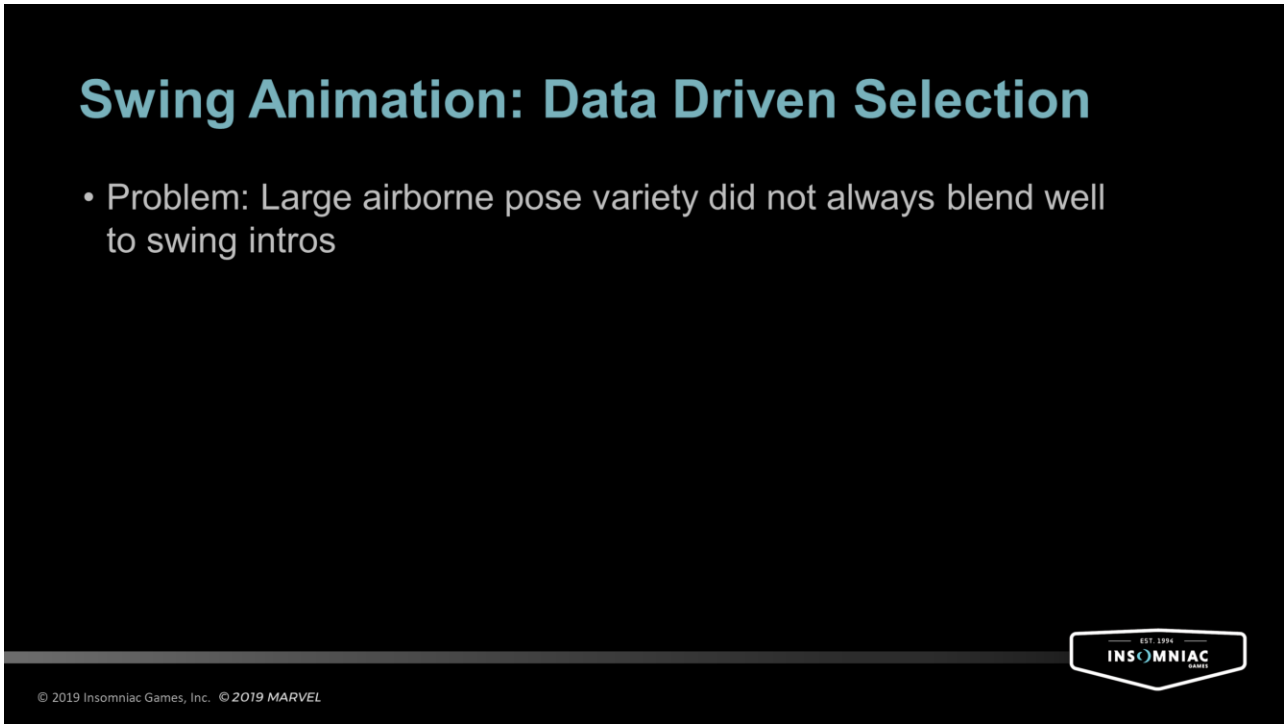


- By breaking the swing animation into pieces it allowed us to have more variety in animation construction.
  - We were able to introduce custom kicks and spins with custom upswings and hard “catch” variations of the downswing all of which could be

played with or without one another.



- This new iteration help us solve our major problems by allowing us to create a lot more animation variety
- It also helped us eliminate the animation scaling from the most notable portions of the swing

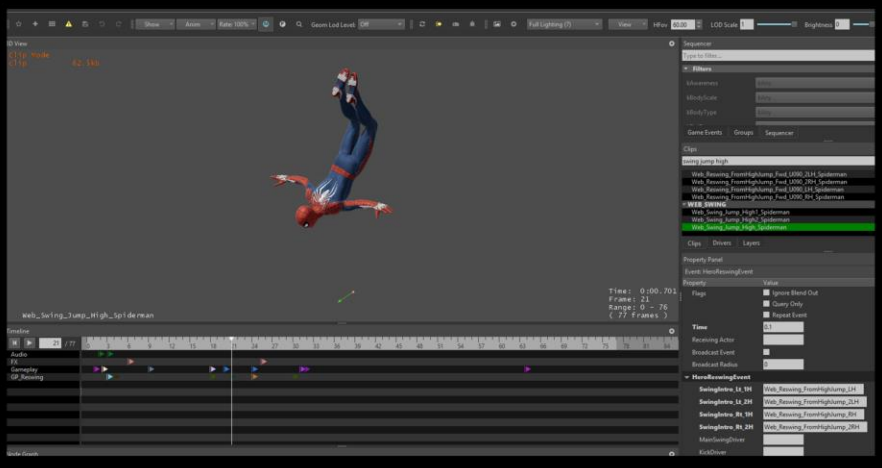


- One final problem worth noting was that with all the various moves that Spider-Man can do he would often end up in poses that did not play nicely with our swing intro animations.
  - We needed to either wait until the character got to a better pose or be

okay with some really bad looking blends.

- To solve this, animation wanted to add more variety for swing intros but programming didn’t want to manage each of those clips manually.

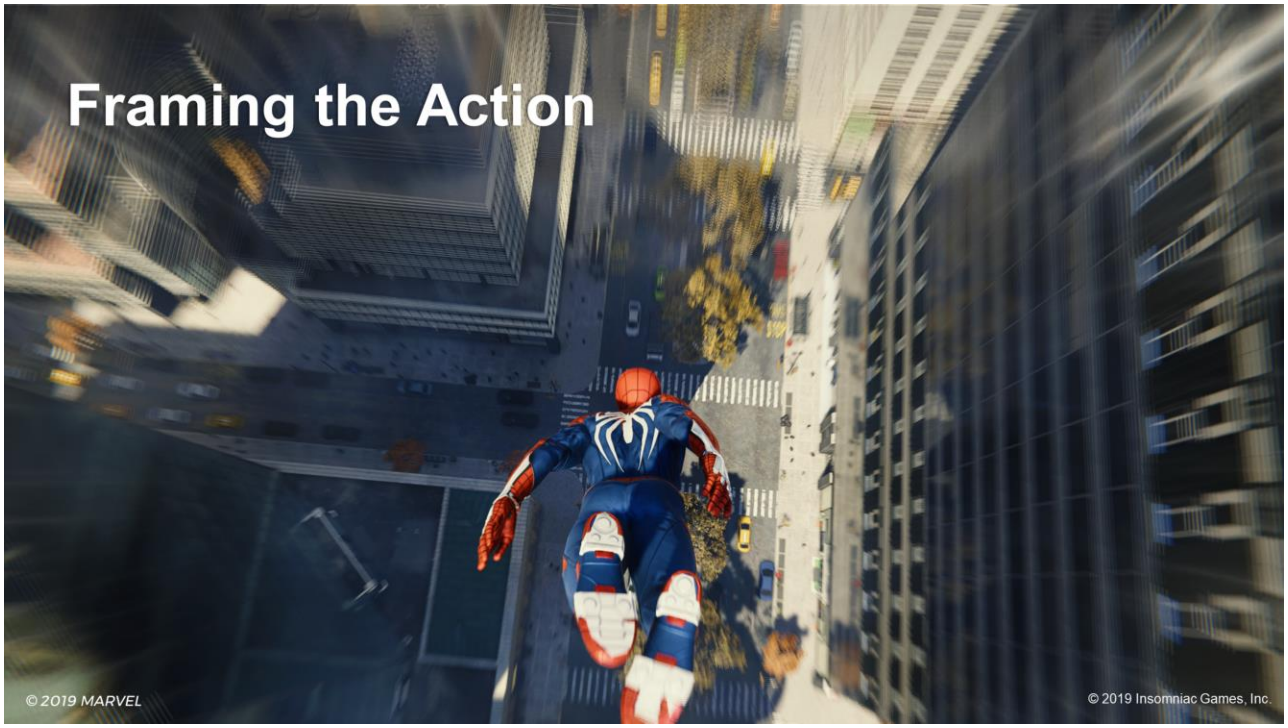
Swing Animation: Data Driven Selection



- To push the work to the content creators, we added a data-driven setup that allowed animators to tell the game what the next animation would be from the current time in the active animation
  - They would mark up sections of the jump animations with data

that would point to custom sling animations.

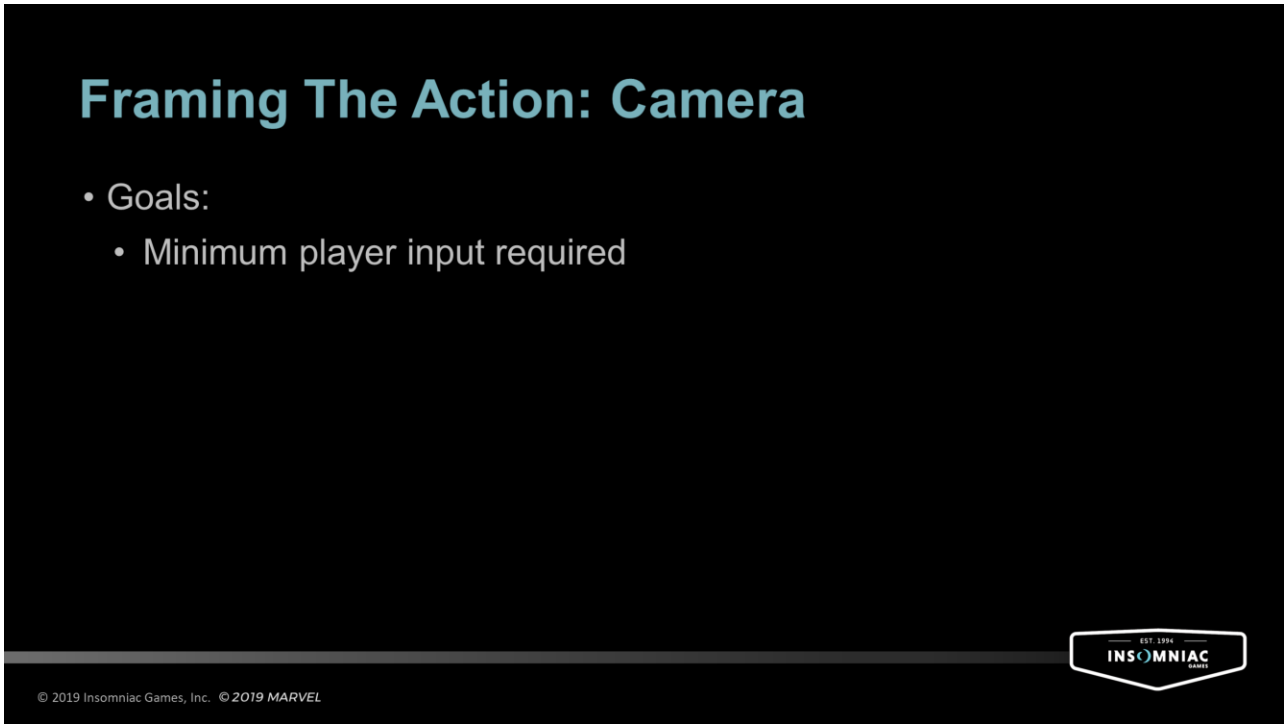
- This allowed animators to cleanly get out of acrobatic poses with custom slings making the traversal feel more seamless and the character feel more dynamic.



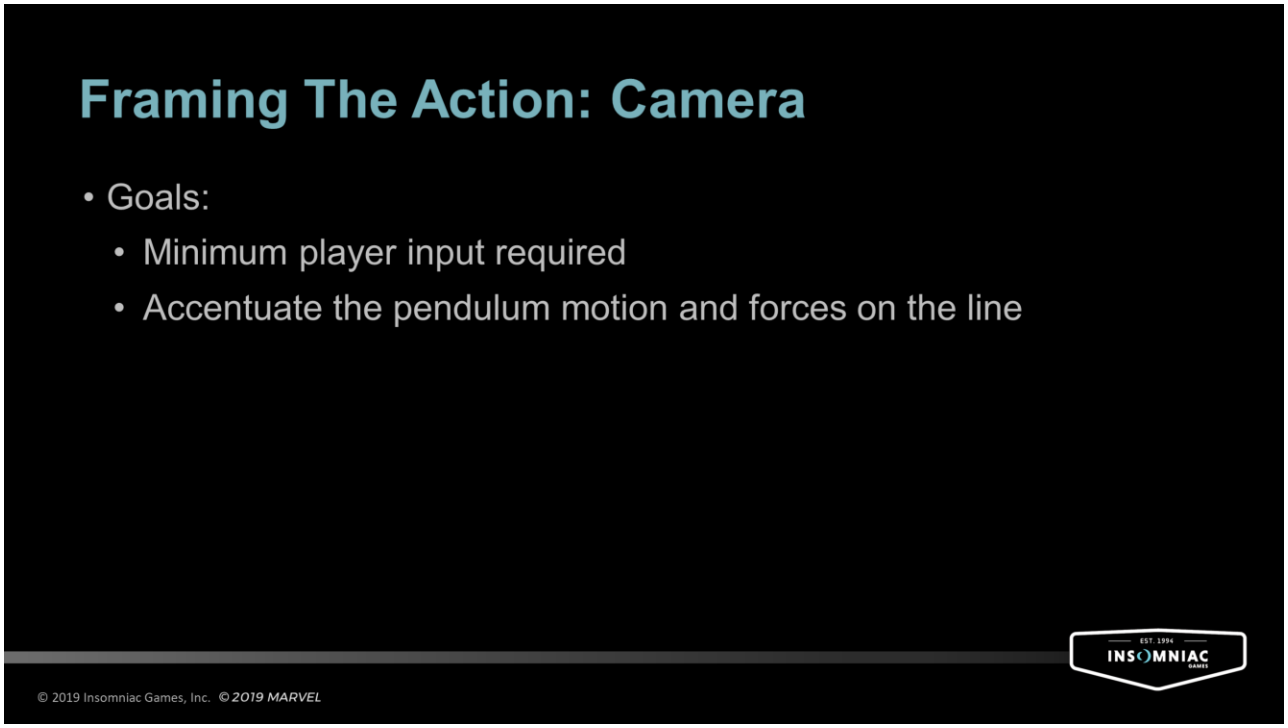
Moving on from animation, I want to talk a bit about our approach to the traversal camera. We knew from previous games that good motion and animation only goes so far in translating the experience to the player. The camera is often the most important aspect in conveying

elements of speed and translation in a way that gives the player a visceral connection to the action on screen.

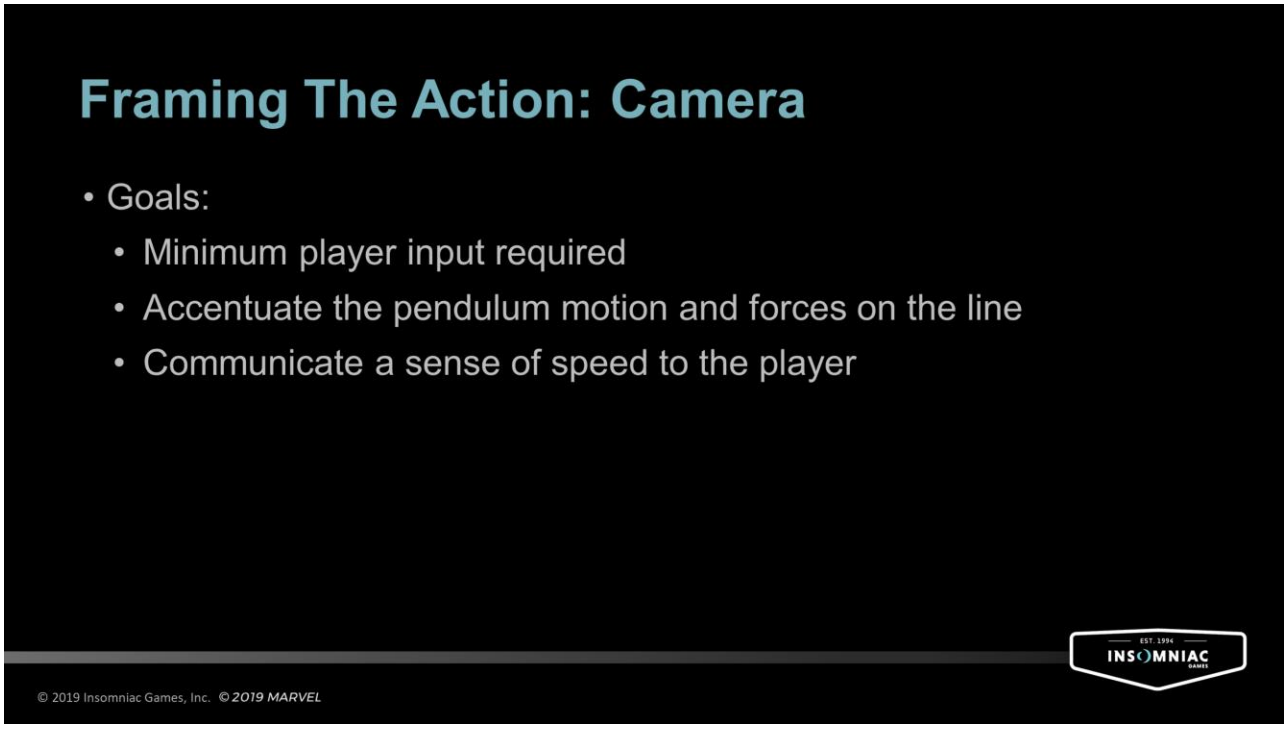
When we started talking about our top-level goals for the traversal camera, we came up with three main items that we found important after early prototyping with a mostly static camera



- First, we wanted to minimize the amount of player input that was required.
  - This is because the player would need to be engaging with the face buttons frequently and could not afford constant camera management.



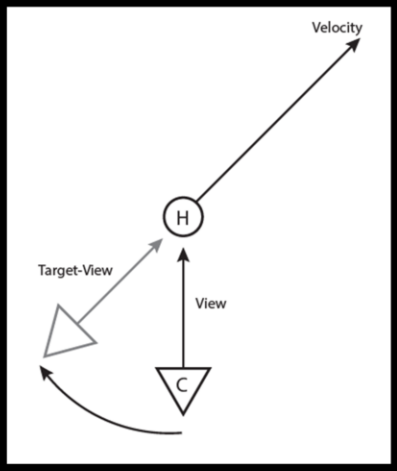
- Next we needed to accentuate the pendulum motion and forces of the line.



- Finally, we wanted to communicate a sense of speed to the player

Framing The Action: Follow Camera Version 1.0

- Camera target view matches hero velocity

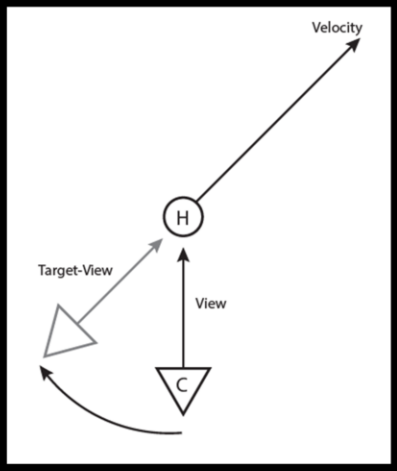


- We have a pretty long history of third person games at Insomniac and have generally started with a follow camera model that tries to have the camera looking in the same direction as the hero's velocity. While successful in other contexts, it presented a few problems for Spider-

Man.

Framing The Action: Follow Camera Version 1.0

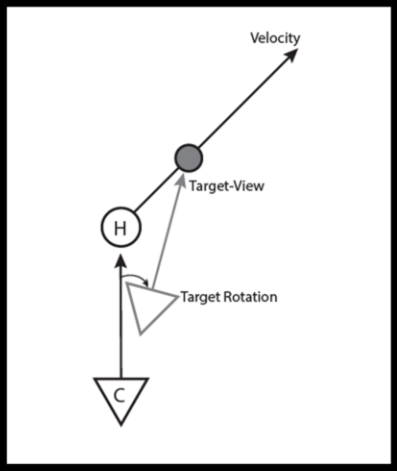
- Camera target view matches hero velocity
- Problem: Too noisy relative to velocity changes



- The biggest issue was that when tuned to effectively show you where you were going, it was extremely susceptible to sudden changes in velocity causing large changes in view direction.

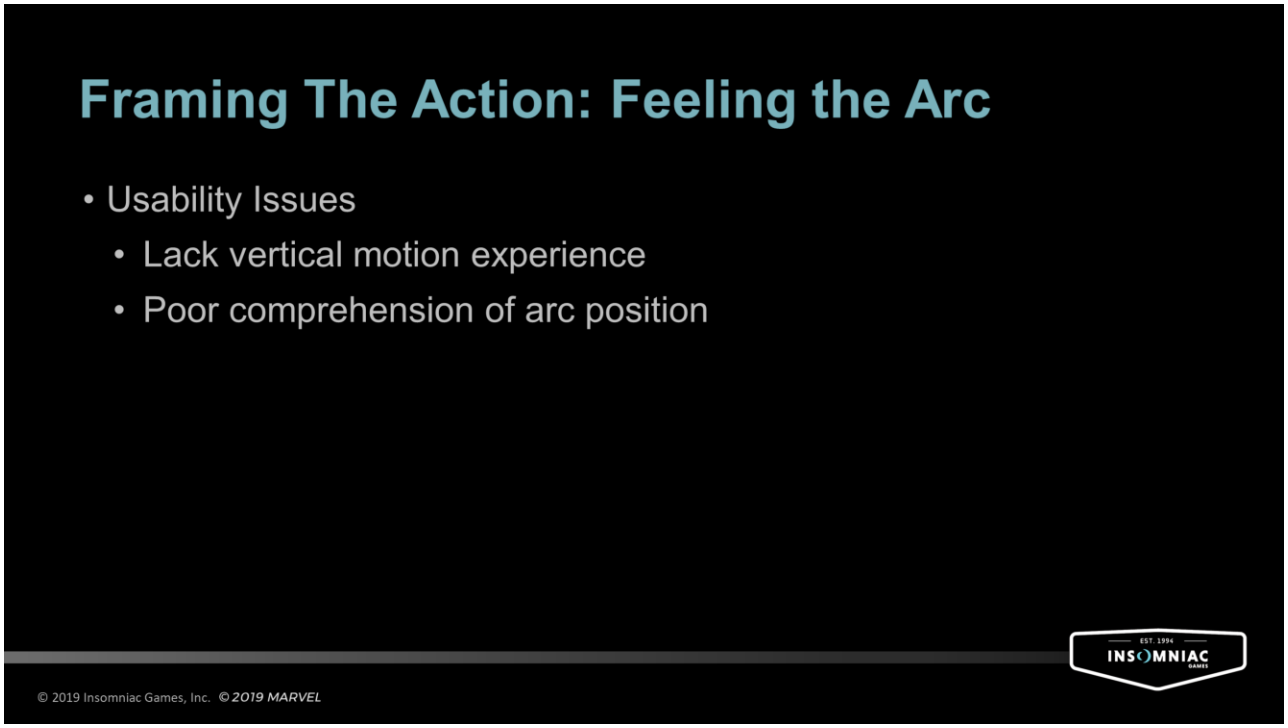
Framing The Action: Follow Camera Version 1.0

- Generate ideal from vector to predicted hero position
  - Acceleration for smoothing
  - Less susceptible to sudden change
  - Scales with player speed



- In order to fix our problem, we moved to a model based on the player “dragging” the camera around.
  - This was done by predicting the player’s position based on their current speed and using it to calculate how that would drag the camera boom.

- This ended up be much less noisy with velocity changes as the frame-to-frame delta was generally a lot smaller.



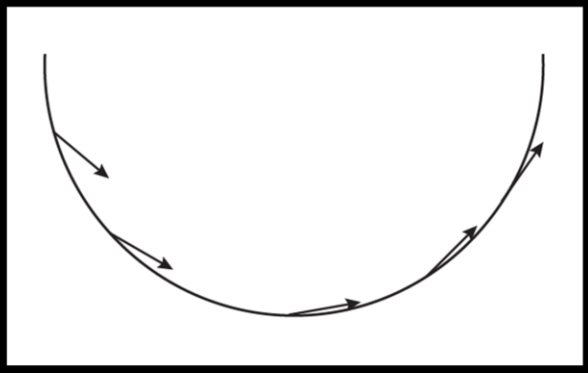
- With the camera now following us more appropriately I want to talk about how we accentuate the pendulum motion of the player through camera motion.
  - In addition to just feeling better, this also helped us address two major usability issues that a more

static camera was presenting

- First, as stated before, players were not really noticing the full extent of vertical translation the character was actually performing.
- Players were also presenting a poor level of comprehension of where they were in the swing arc.
- They were frequently hanging on too long, releasing high, and losing all their forward momentum.

Framing The Action: Feeling the Arc

- Camera Pitch
  - Position on arc drives target camera pitch
  - Flatten on down-swing
  - Over-pitch on up-swing



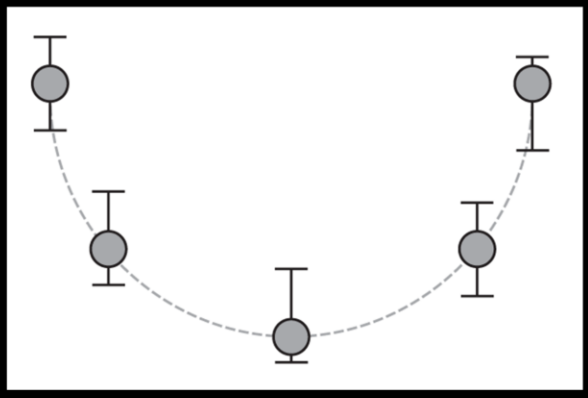
- The first thing we manipulate to help sell the swing arc is the camera's pitch.
  - We try and roughly match the swing-arcs pitch in order to feel that motion.
  - As you can see here, we want to keep the pitch a little flattened off

on the down-swing so that it’s still enough to feel but not so much that players lose sight of the horizon line.

- After the trough we then exaggerate the pitch early in the up-swing in order to communicate rise more effectively.

Framing The Action: Feeling the Arc

- Vertical Offset
  - Lower character screen position at trough
  - Raise character screen position through up-swing



- The next element we added to help players feel the swing arc more was to move the character's position in screen space down and then up over the course of the swing.
- This created a real-world physical tracking response from the player that would reinforce the in-game

motion.

Framing The Action: Feeling the Turn

- Roll camera during turns
  - Driven by line angle, turn speed

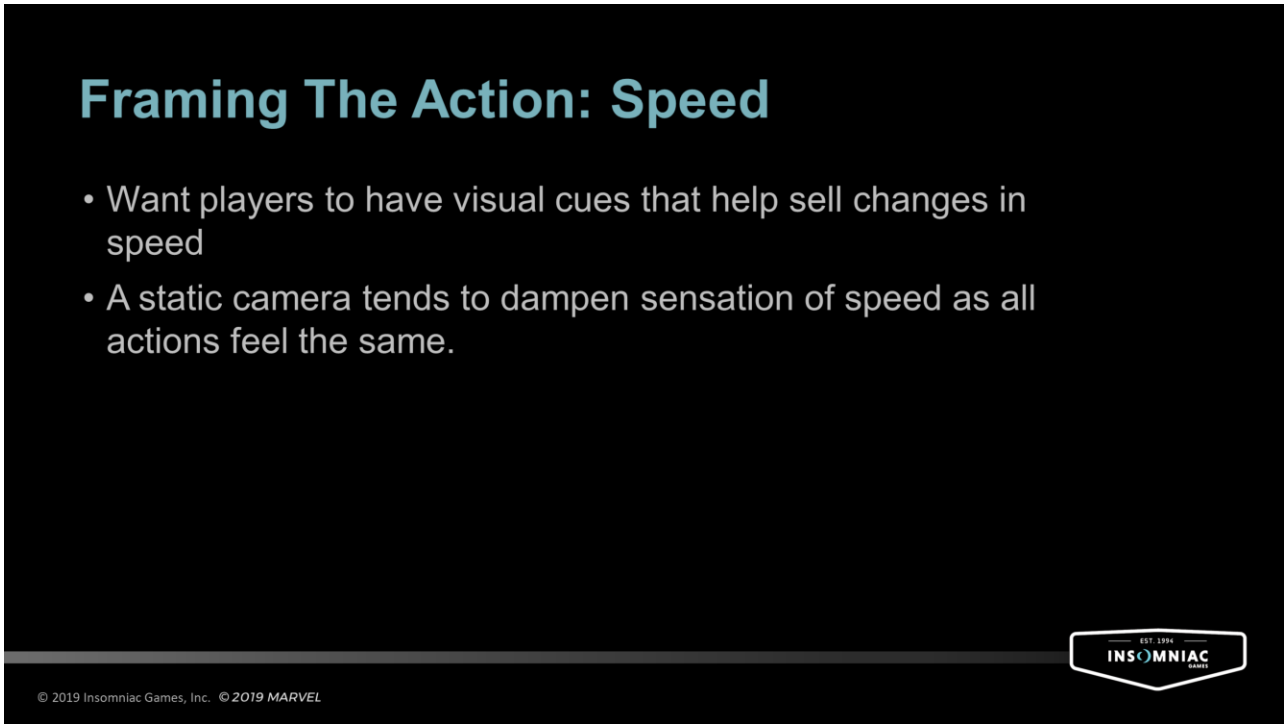
- When it came to turning we wanted to get the player to feel like they were whipping around a corner at high speed and bring the same sensations you might have in a turning car.
  - To get this affect we apply a little bit of roll to the camera when

turning and scale it based on the line angle and turn speed

Framing The Action: Feeling the Turn

- Roll camera during turns
  - Driven by line angle, turn speed
- Can't push too far!
  - Player motion sickness a real concern
  - Option to scale down/off

- However, we had to be careful not to push it too far due to motion sickness concerns.



- Now that we have players feeling the translation in traversal, we need to make sure that the speed is being properly conveyed.
  - This means we needed to layer as many visual cues as possible to help sell not just the current speed but changes in speed.

- Probably the most impactful thing to sensing speed through the world is the field of view.
  - A wider field of view elongates objects in front of the player making it feel like you are covering more ground
  - It also brings more nearby,

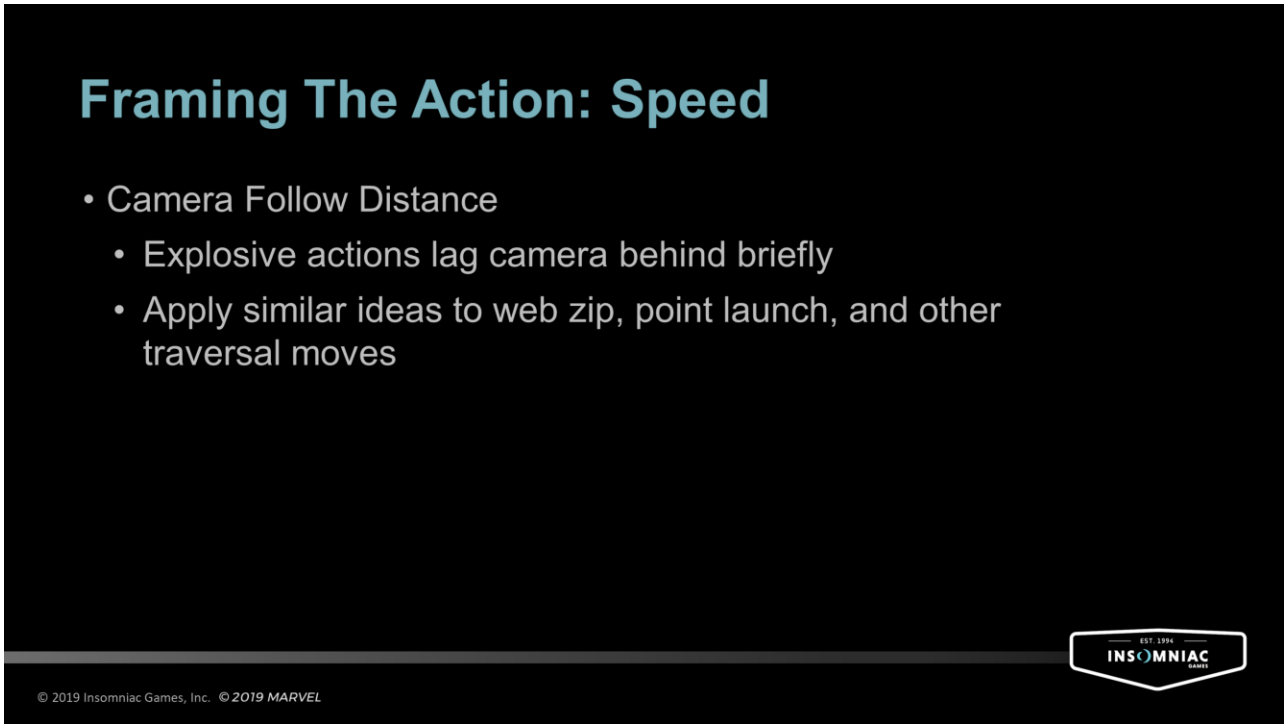
peripheral objects into view that will move very fast by the player.

- In order to really sell the changes in speed we are constantly updating the game’s field of view while traversing.

- To deal with sudden speed changes, we rely more on adjusting the camera's follow distance.

- We have certain explosive moves create a spring action on the camera so that it falls back before blending closer again
  - This gives those moves a much bigger sense of impact, like the character is boosting away suddenly, leaving the camera

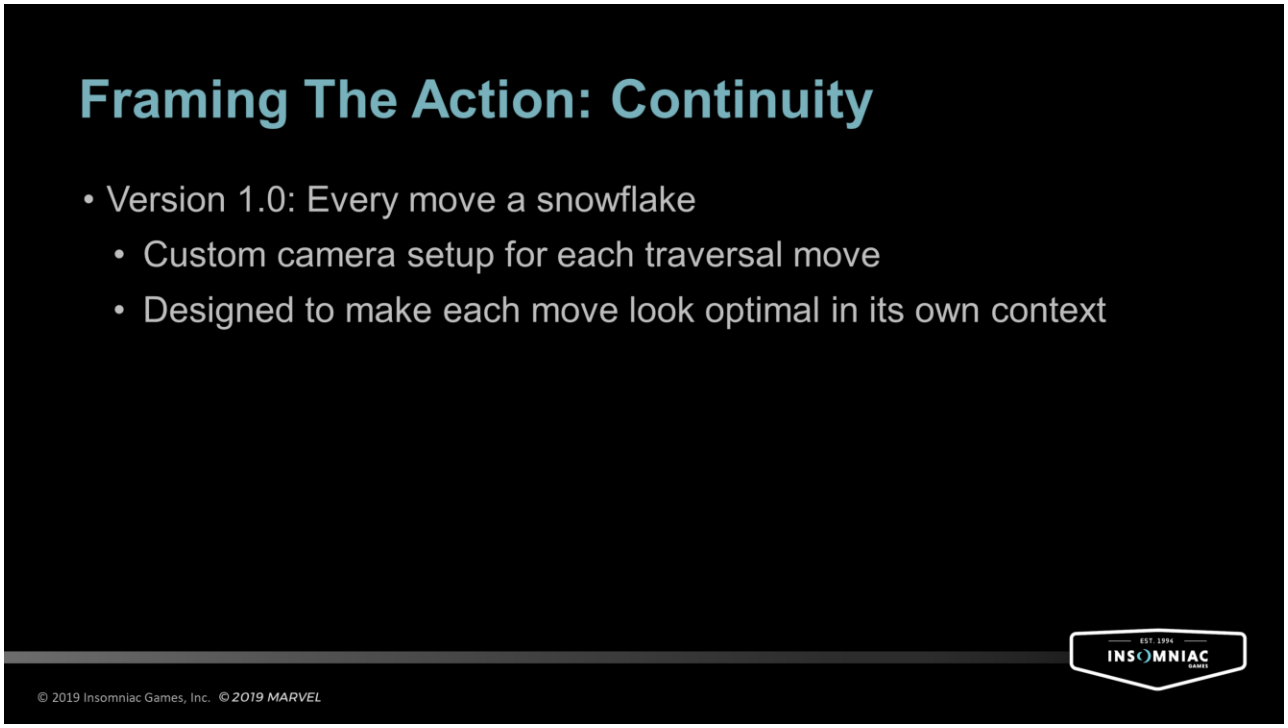
behind.



- We can apply similar ideas to other moves like web zip and point launch to add impact.

- [BONUS INFO]
- We can also use camera distance to sell anticipation.
    - When swinging, the camera will slowly close the distance to the player. Then, on a jump release, the camera will fall back suddenly creating the overall sensation that the character really threw himself off the line with power.

- Here you can see that all come together to show the sense of speed
  - [ADVANCE]
  - There is some debug draw on the side to give you a better idea of how the field-of-view and follow distance are changing with the action



- The final bit of camera development I want to talk about is related to how the camera is handled as we perform a variety of traversal moves.
- Our first version of the traversal camera treated each move as a special, unique item.
    - This was driven by a desire to

make every move as cinematic as possible with the thought that stringing together a bunch of cool cinematic moves would look extra cinematic.

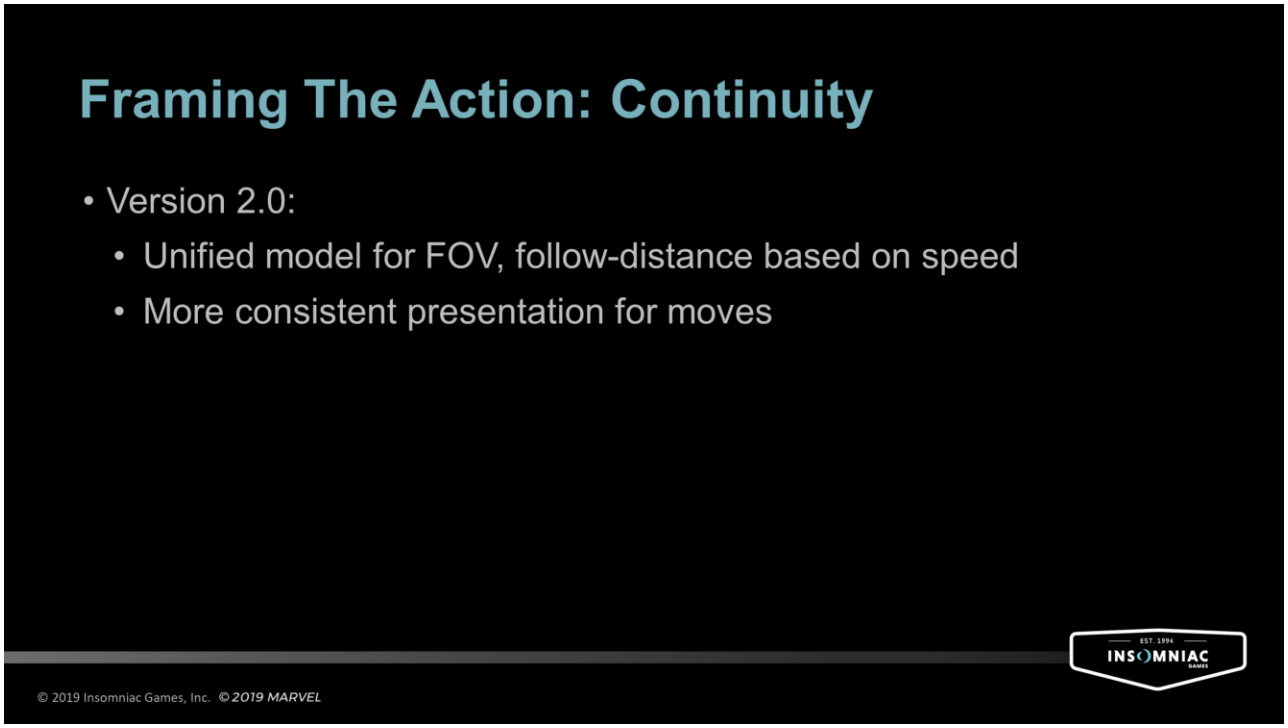
However, this presented a few problems…

- This is a video from an early vertical slice demo that will present some of the issues with how the “every move gets a special camera” method works.

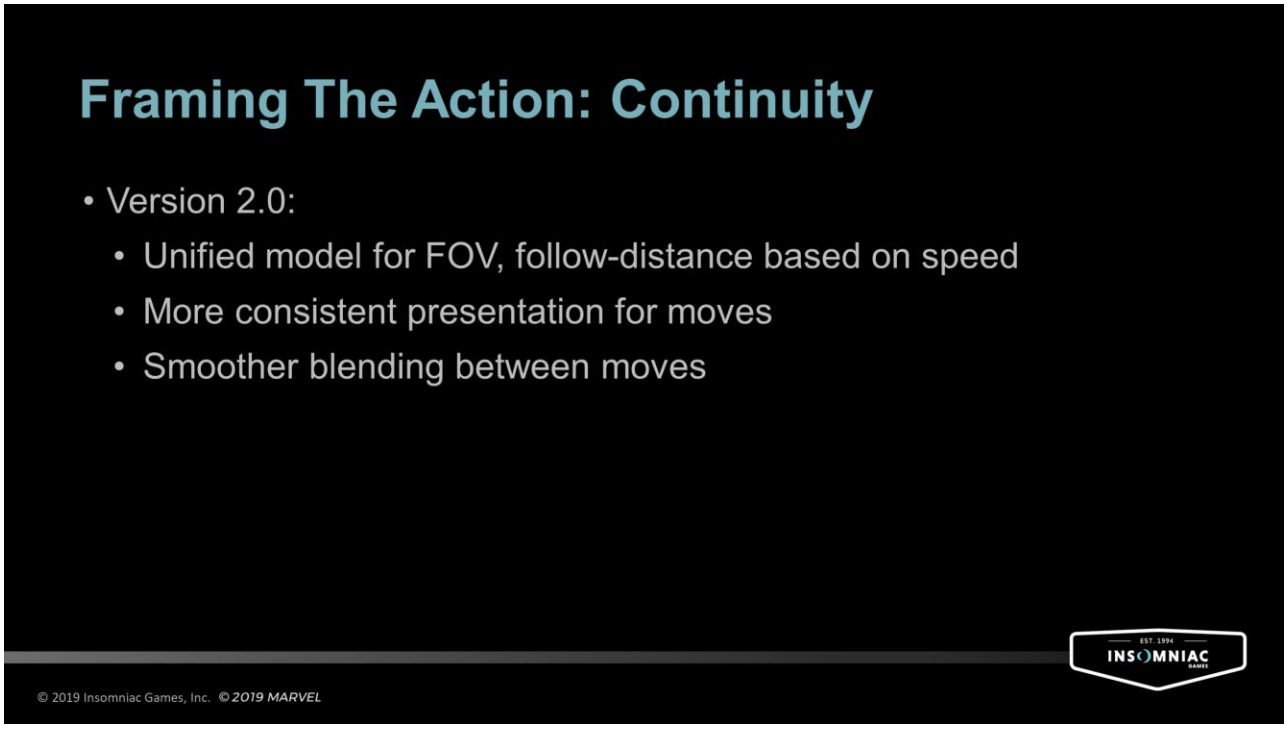
- [PLAY VIDEO]

- You’ll see that going from move-to-move would cause a change in framing and FOV that could muddle the perception of speed.
- Rapid sequencing of moves would also create a very noisy relationship between the world, character, and view.

- So, how did we get better?



- We started with a more unified model for all traversal moves that had speed-driven FOV and follow distances that would stay consistent across multiple different moves.



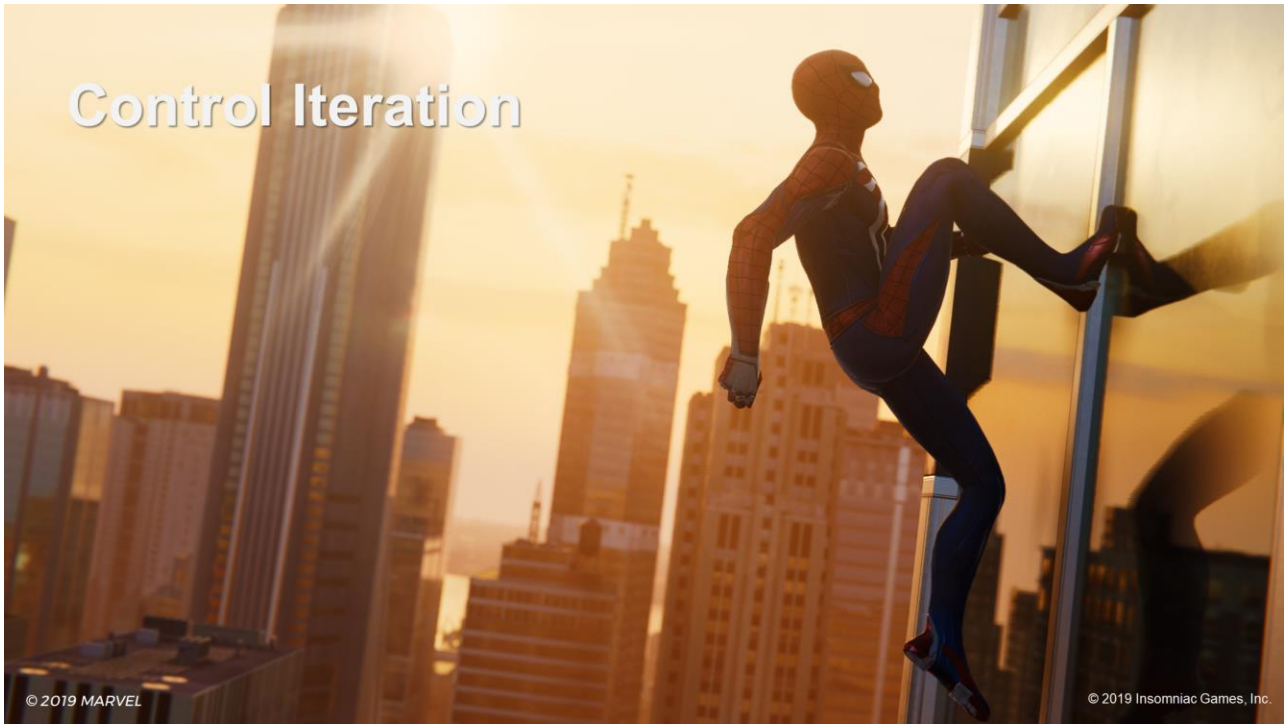
- We also smoothed out any adjustments we were doing to offsets and made those changes less aggressive.

[PLAY VIDEO]

- From those changes we got some pretty good results.
  - You can see here that there is a much more consistent perception of speed when moving from one move to another.
  - The camera ends up being a lot more predictable to the player and one move flows into the next without feeling like there is a

major state change for the player to adjust to.

- This proved a lot more fun and predictable to players and gave an overall better experience.



- As we started to get a full set of traversal moves in the game and playing with them more, both internally and through usability, we found that, unsurprisingly, our first version was rarely good enough.
- This lead to a lot of feedback that motion felt stiff and imprecise.

- Players weren’t able to navigate as well as they wanted, many moves weren’t matching expectations, and they were unable to land where they wanted.

- So, we started re-visiting them.

Control Iteration: Wall Run

- Version 1.0:
  - Pure horizontal or vertical wall running only
  - Moving parallel to wall biased horizontal entry selection
  - Moving towards wall biased input entry selection

- Starting with wall run, our first pass was set up to be locked as either horizontal or vertical with no in between.
- Entry was based on a combination of character velocity, input direction, and the wall's facing with a heavy bias towards maintaining the current

momentum and direction.

Control Iteration: Wall Run

- Problems
  - Poor mapping of player expectation to run direction
  - Camera direction versus wall facing provided poor input map

- The single biggest problem we had was that wall run entry direction was poorly mapped to player expectations.
  - They never really understood the concept of momentum driving the entry direction and overwhelming their input

- This was made even worse because there was no ability to switch from horizontal to vertical to correct problems

Control Iteration: Wall Run

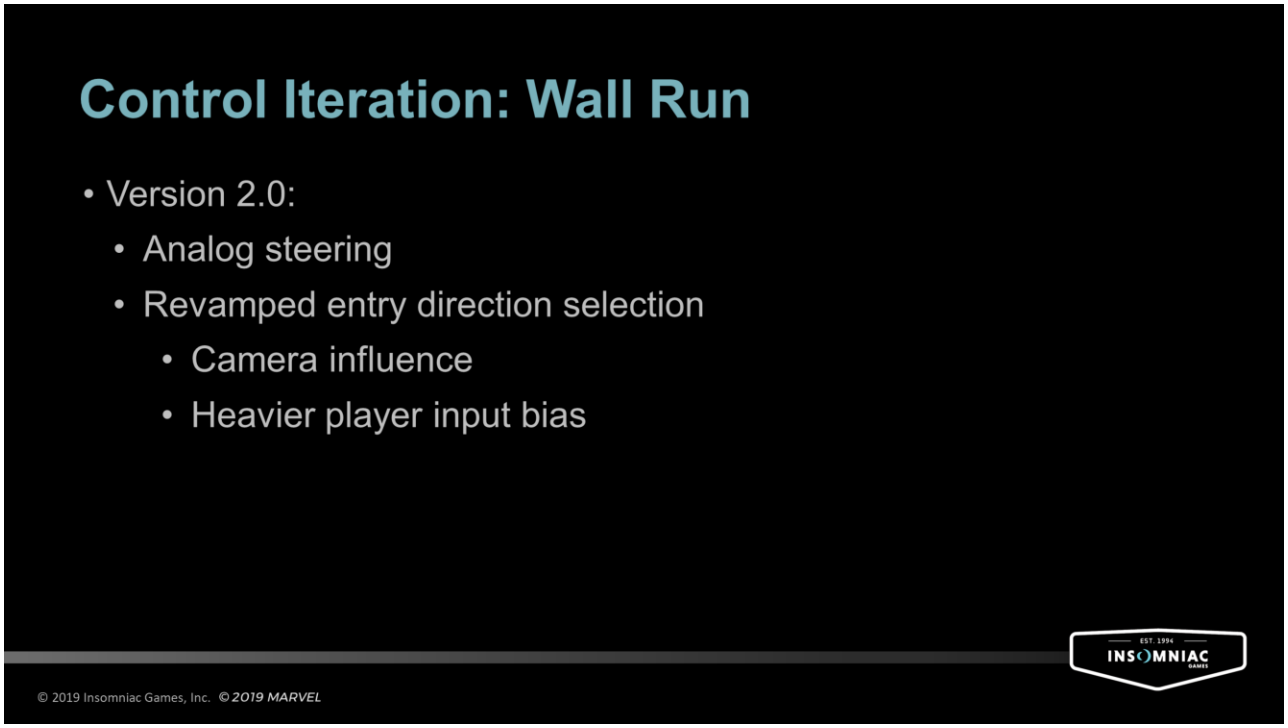
- Problems
  - Poor mapping of player expectation to run direction.
  - Camera direction versus wall facing provided poor input map

- On top of that, camera direction versus a vertical wall was making input-to-entry mapping even more confusing.
  - In this image is up on the stick supposed to mean “go forward” or “go up”?
  - Players would often try and

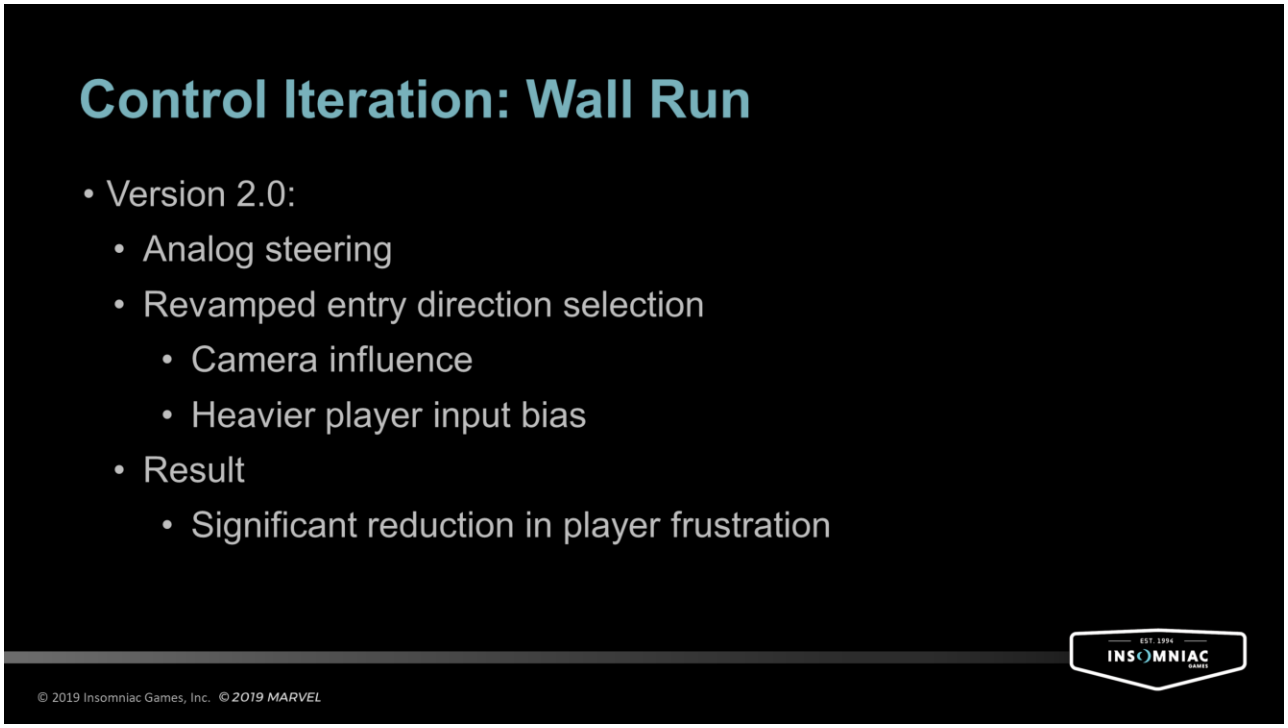
compensate for this by pointing the camera where they wanted to go but the system wasn’t paying attention to the camera.

- So, with our big problems identified, we set out to make some changes.

- The first thing we did was add analog steering. This created more in-between values for the entry direction as well as giving players the ability to adjust on the fly from an entry direction they weren't happy with



- Then we revisited our entry direction process.
  - We added a camera-driven influence to help translate player intent to in-game results.
  - We then amped up the effect of player input on direction over momentum.



- These combined to result in a significant reduction in player frustration.

Control Iteration: In-Air Control

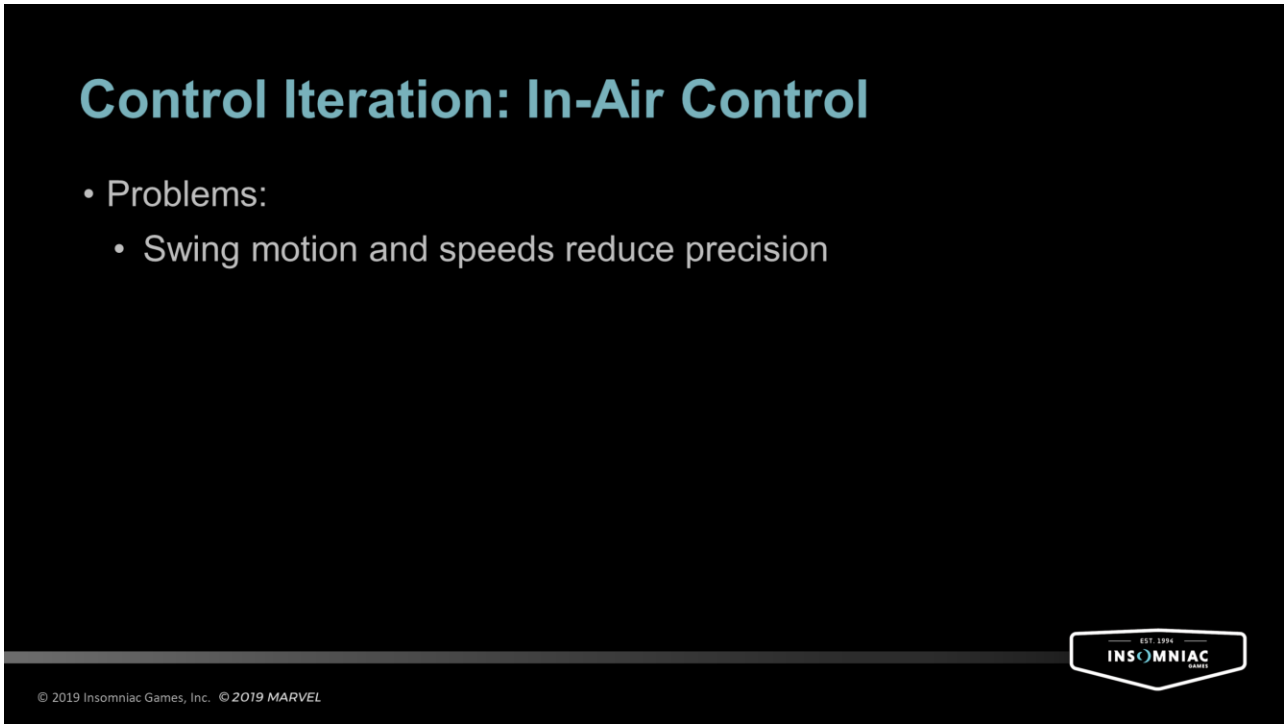
- Version 1.0:
  - Heavy momentum bias
  - Stiff steering
  - Consequential swing exits

- For in-air control, when the player is off the line and airborne, our initial approach was to again heavily favor momentum.
  - This meant that in-air steering was fairly stiff and players needed to rely on web moves like swing and web zip for rapid directional

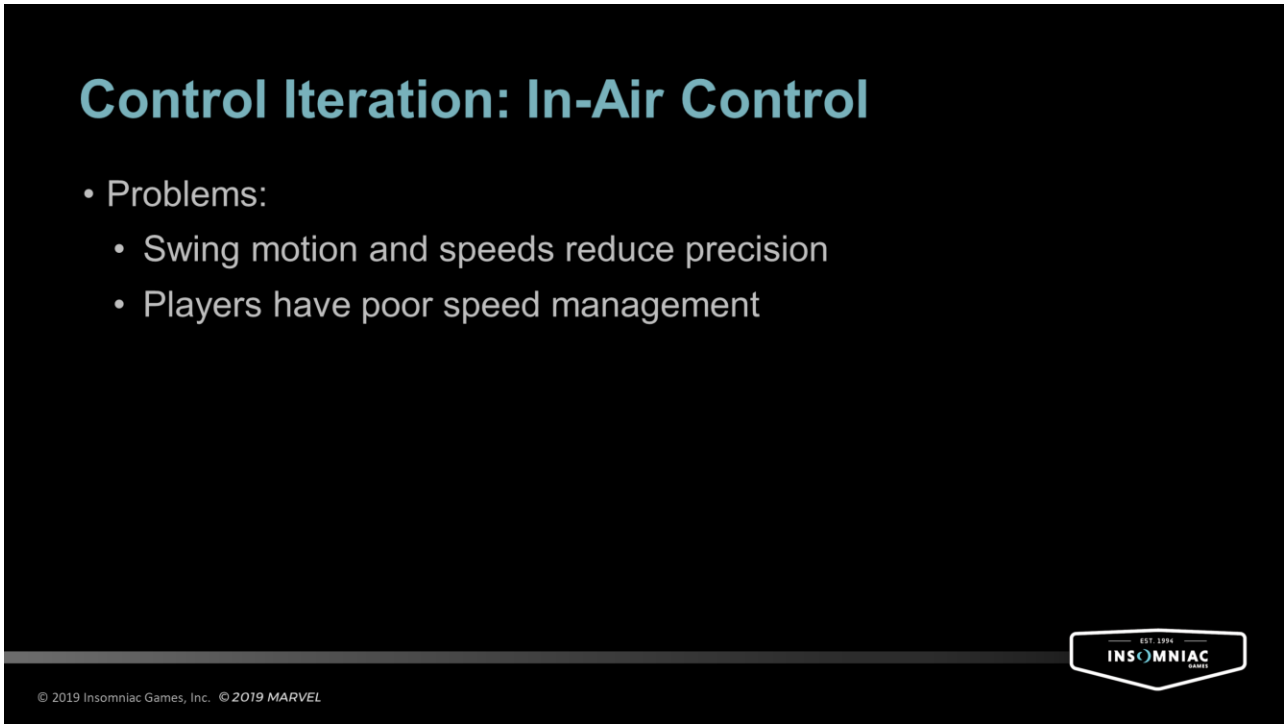
change.

- The idea was that it would make your swing releases more meaningful and consequential.

- However, this model presented a number of problems in playtesting.

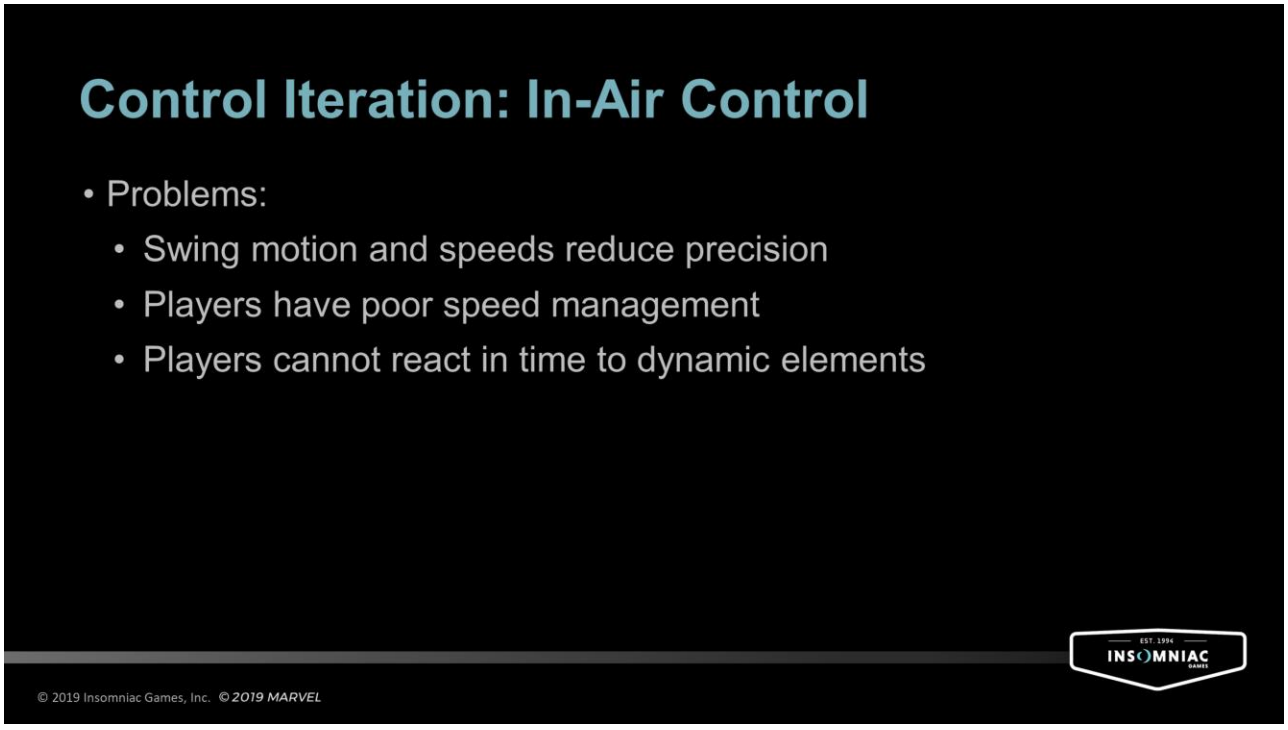


- First, swing speeds amplify small differences in angle over large distances.
- This made it almost impossible to be precise.



- In addition, players are just really terrible at thinking ahead to manage their speed.
  - While there are tools to slow down, they will generally go as fast as possible to their destination and only then try and stop.

- This usually ends with the player fumbling around, orbiting their destination as they try and get closer.



- Finally, players were having a lot of difficulty reacting quickly to changes in situation or geometry.

Control Iteration: In-Air Control

- Version 2.0:
  - Increase in-air turn speed

- In order to help solve these problems we had to put more control back in the players hands and let go of some of our strict momentum-based concepts.
  - [ADVANCE] To start, we significantly increased the in-air turn speeds.
    - This immediately helped players

react much more quickly to dynamic elements.

- It also allowed people to more effectively set up their next move, increasing their ability to chain moves

Control Iteration: In-Air Control

- Version 2.0:
  - Increase in-air turn speed
  - Increase drag on stick-back input

- Later on we massively increased the amount of drag applied to the player when pulling back on the stick.
  - This let players feel a lot more precise when reaching their destination because they could go from 60 to 0 in a very short time.

- Overall, this lead to a lot fewer instances of players fumbling around their destination and let people feel like they could stick the landing.

Control Iteration: Point Launch

- Version 1.0
  - Wanted tactile world interaction
  - Flying through objects with web tunnel

- Next we needed to address our more specific world interaction moves.
  - When we started making the game we knew we wanted to make the city feel like a big part of traversal and Spider-Man's interaction with the world needed to be tangible. However, while swing and web zip

required geometry to work you were still mid-air when using them.

- We wanted something more tactile.
- Our first version of this was zipping through the legs of a water tower with our web tunnel move.
  - This was very cinematic from a presentation standpoint and definitely fulfilled the fantasy of interacting with specific objects bit it had a few drawbacks

Control Iteration: Point Launch

- Version 1.0
  - Wanted tactile world interaction
  - Flying through objects with web tunnel
- Problems
  - Rigid directionality
  - Poor entry engagement

- The first problem is that the move had pretty rigid directionality. There wasn't a lot of room for players to modify their direction going through the objects.
- In addition, the entry angles were rather limited. You could only engage the web tunnel from a limited angle

range before it started to look and feel really strange to get sucked into the tunnel.

Control Iteration: Point Launch

- Version 2.0
  - Provided world interaction
  - Omni-directional points

- In this case, we recognized that web tunnel had some inherent limitations that we weren't going to be able to break. In addition, we liked having the move as it was for certain situations.
- Instead, we created point launch, a move where you could zip to a point

on top of an object and spring back off it instantly with a big jump.

- We were able to get the desired world interaction we were looking for with web tunnels along with a number of additional benefits.

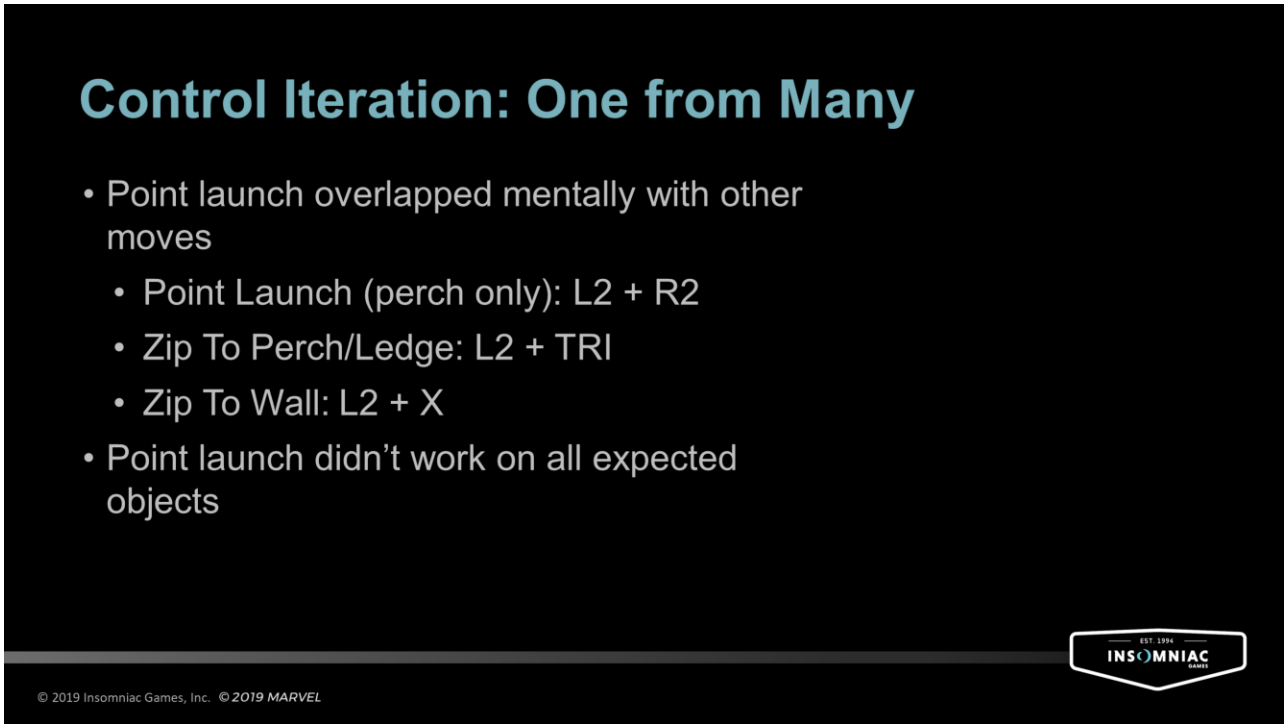
Control Iteration: Point Launch

- Version 2.0
  - Provided world interaction
  - Omni-directional points
- Benefits
  - Simpler usability
  - Easier placement
  - Increased engagement

- First, The player could enter and exit a point launch from any direction
- Next, because we just needed a point, the opportunity for placement in the world increased dramatically.

- Once point launch was implemented, we started to notice that we had a lot of traversal moves that we were treating as unique moves but to players felt like the same action.
- In this case, that action was “go to a thing”.

- As you can see, we had separate, unique button combinations for zipping to a perch, point launching, and zipping to a wall



- To make matters worse, point launch only worked on perches while a similar move worked on perches AND ledges
- Ultimately, this led to a lot of problems with players' ability to mentally map a variety of controls to what felt like the same action.

Control Iteration: One from Many

- Solution: Merge all moves into a single input of L2 + R2

- Our solution then was to merge them all into a single input space using L2 and R2
  - This helped ensure that the button-to-action mapping in players' brains was more 1-to-1.
  - However, it wasn't just a matter of merging buttons, it required us to

adapt some of the moves as well.

Control Iteration: One from Many

- Solution: Merge all moves into a single input of L2 + R2
- Adjustments:
  - Point launch on all edges
  - Require aim mode for zip-to-wall

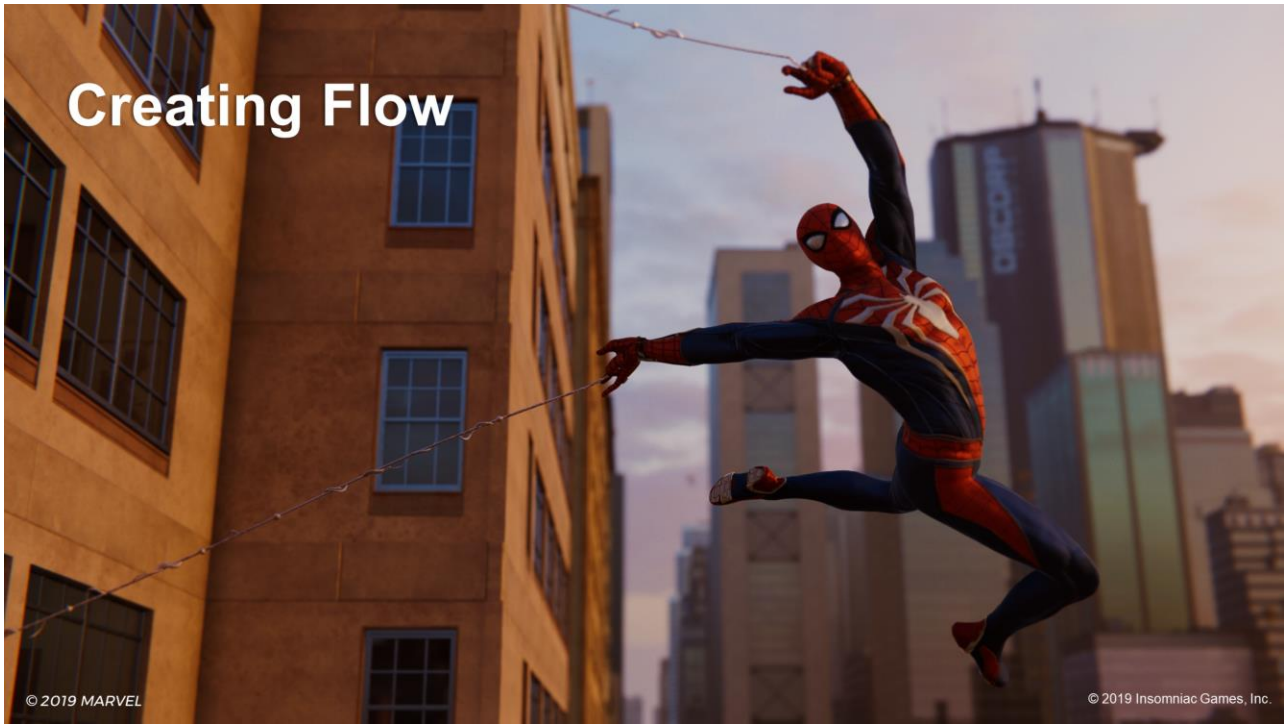
- First, we expanded point launch to work from all edges and merged it with zip-to-perch.
- We then moved zip-to-wall to be an aim-mode only action so that its targeting didn’t conflict with the other moves it was sharing buttons with.

- So, after iterating on our controls by doing things like changing buttons and adding moves, there are a few major takeaways that I feel are important.

- First, if player expectations and system design are not aligned then controls fail and player experience drops.
    - We spent a lot of time giving long-winded design reasons why our controls were the way they started but ultimately players will

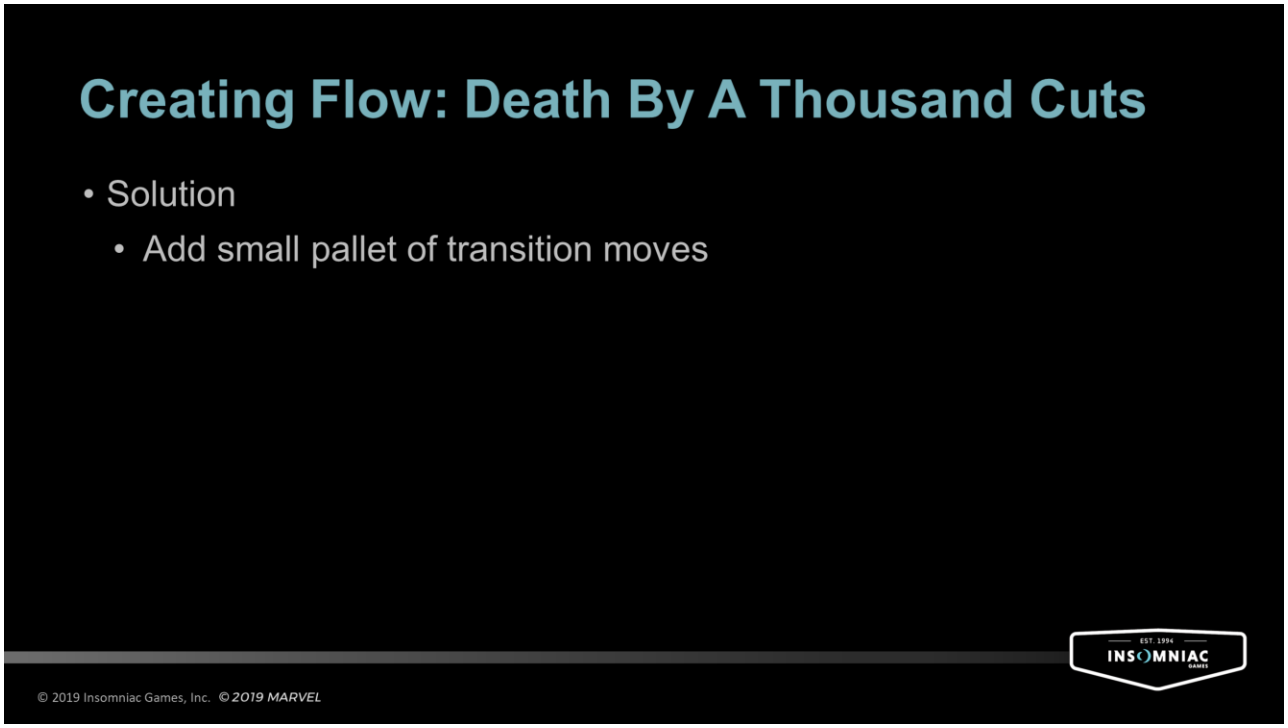
never get the most out of a system they have trouble using.

- The next lesson would be that trying to add analog input and behavior to our moves helped add depth, utility, freedom, and flow to our moves.

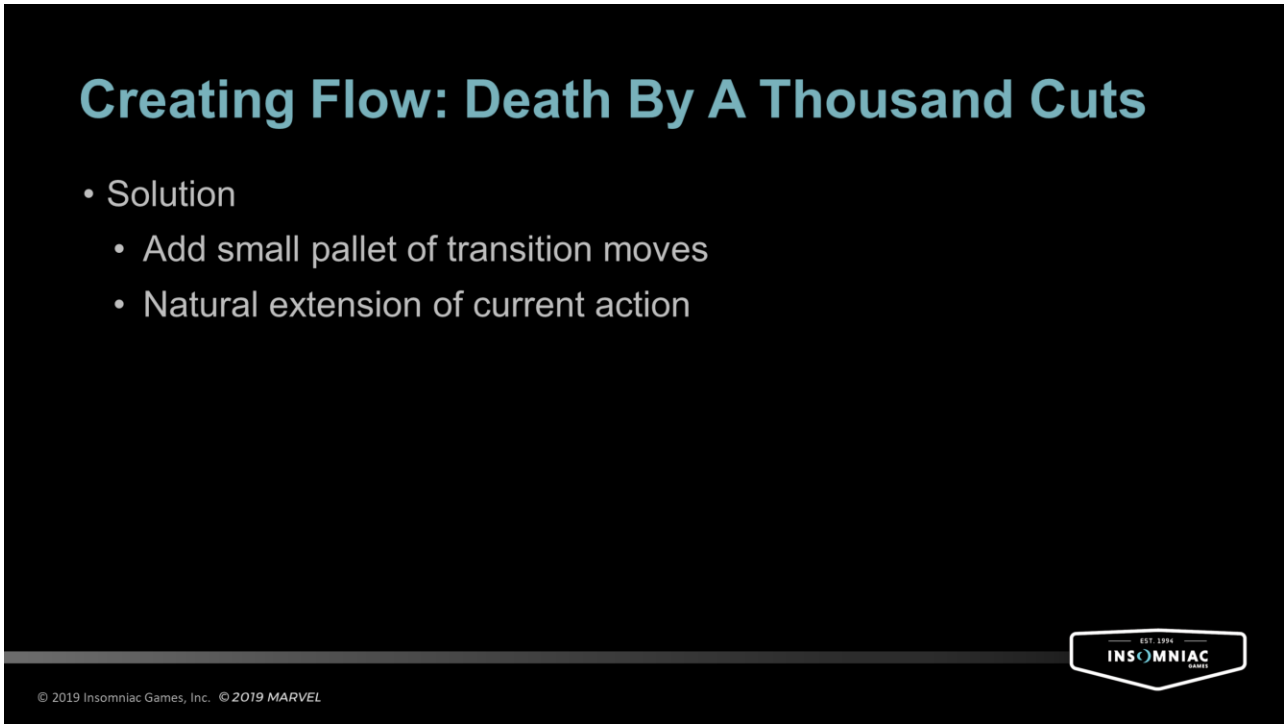


- As we were getting towards the end of production and our core mechanics were all in place, we started to drive down to the next level of usability feedback.
  - We found that players were still feeling like the experience had a somewhat stuttered rhythm

where it would feel incredibly smooth at times but they were constantly finding little snags to break their momentum



- As we kept gathering information, we found that most of the problems boiled down to a few cases repeated across the city.
  - In response, we created a small pallet of moves that would help maintain flow around those rough edges.

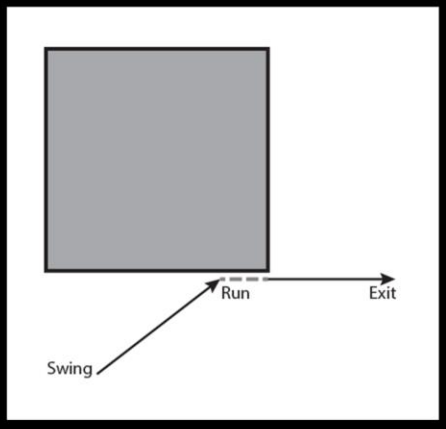


- Many of these moves were designed to be a natural extension from the character's current action and so required no input.
  - We wanted the player to feel like they were expertly executing these moves that would often require unnatural

reflexes to execute manually.

Creating Flow: Corners

- Clipped corners would unexpectedly redirect velocity

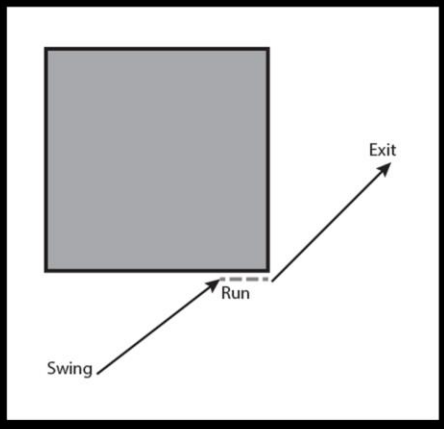


- My first example shows one of our biggest issues: Clipping corners causing a short wall run that would eject the player sideways.
  - This sudden change in direction would often surprise players and frequently create situations that were difficult to recover from as

they were exiting at a high enough velocity to often cross the street before realizing what happened.

Creating Flow: Corners

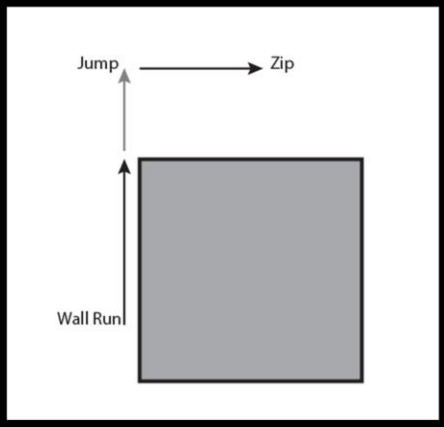
- Clipped corners would unexpectedly redirect velocity



- We fixed this by adding a small move to redirect you parallel to your incoming velocity if you were only wall running for a short time.
- This very quickly got the player back in action with a consistent reference direction.

Creating Flow: Wall Run Crest

- Problem: Players would run vertically off the top of a building and have no forward momentum

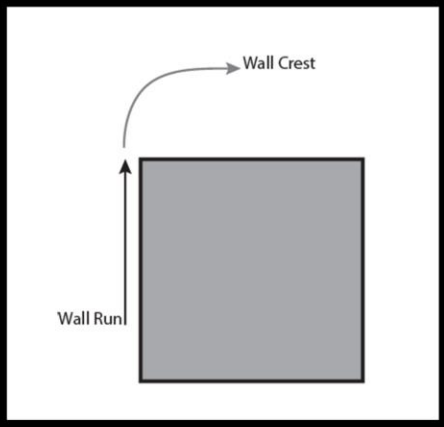


- The next example looks more at a case where players did not have a tool to change direction easily.
  - If a player was wall running up a building and wanted to corner over it, they would have to clear the building and then figure out their next move.

- This lead to a gap in flow while the player processed the new view.

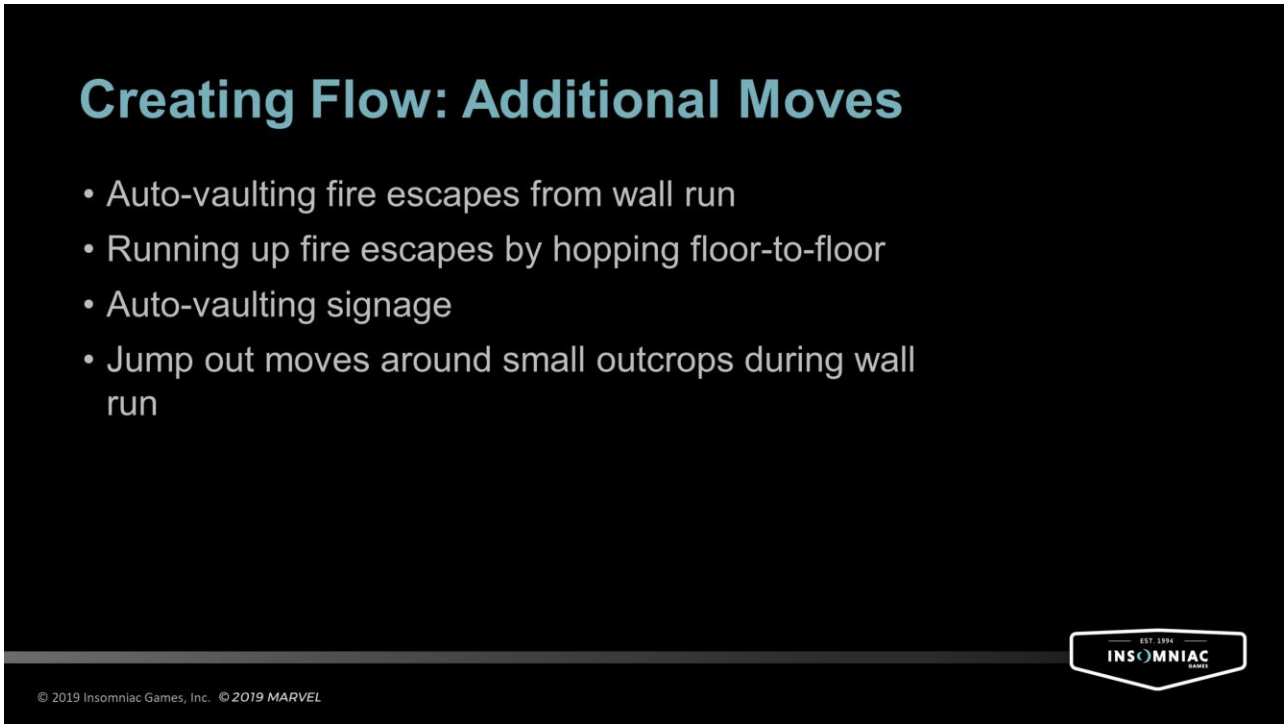
Creating Flow: Wall Run Crest

- Problem: Players would run vertically off the top of a building and have no forward momentum
- Solution: Player-driven move to redirect momentum forward to maintain flow



- To fix this we added a move that players could trigger before clearing the wall to get over the edge and redirect their velocity forward all in one move.
  - This added a small layer of skill and depth while letting players much more efficiently navigate

over buildings.



- A lot of our additional moves are centered around trying to traverse through space near walls and how that conflicts with our art team's desire to have a lot of stuff sticking out of those walls.
  - The biggest factor here was fire escapes. It just didn't feel like New

York without them but they were basically the arch nemesis of the traversal team.

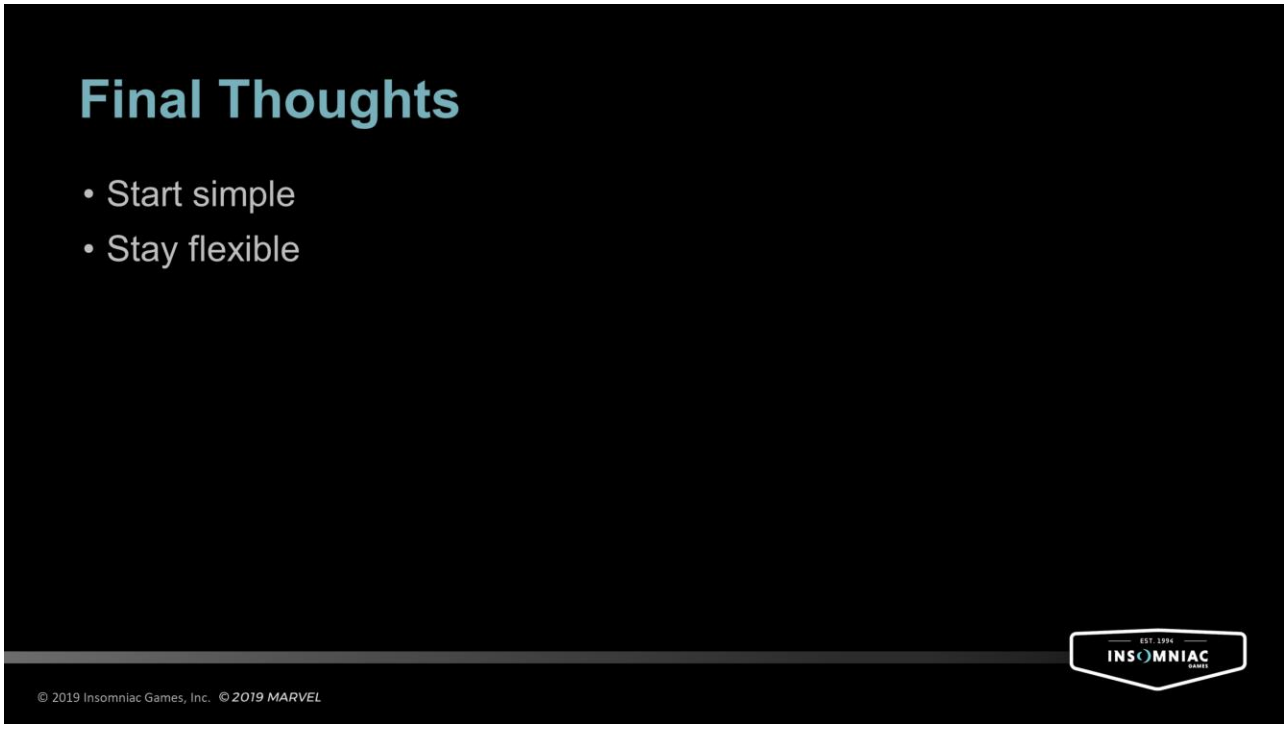
- We ended up making vaulting moves both from the air and from wall run to get around them as well as a completely custom ladder-climbing web move to run up a fire escape stack.
- We also needed to deal with signs hanging off buildings and a lot of small outcroppings in the construction of buildings that were too small to really wall run on but too big to just run over.

- That wraps up the journey we went on creating traversal for Marvel's Spider-Man. We had a lot of fun along the way working with such a great character but there are definitely a few key things I believe any developer can learn from our experience

- First, when creating mechanics, start with a simple implementation and add complexity as you uncover new problems.
    - By keeping things simple early we were able to avoid going too far down a path that our players wouldn't respond to which in

many cases kept us from wasting work.

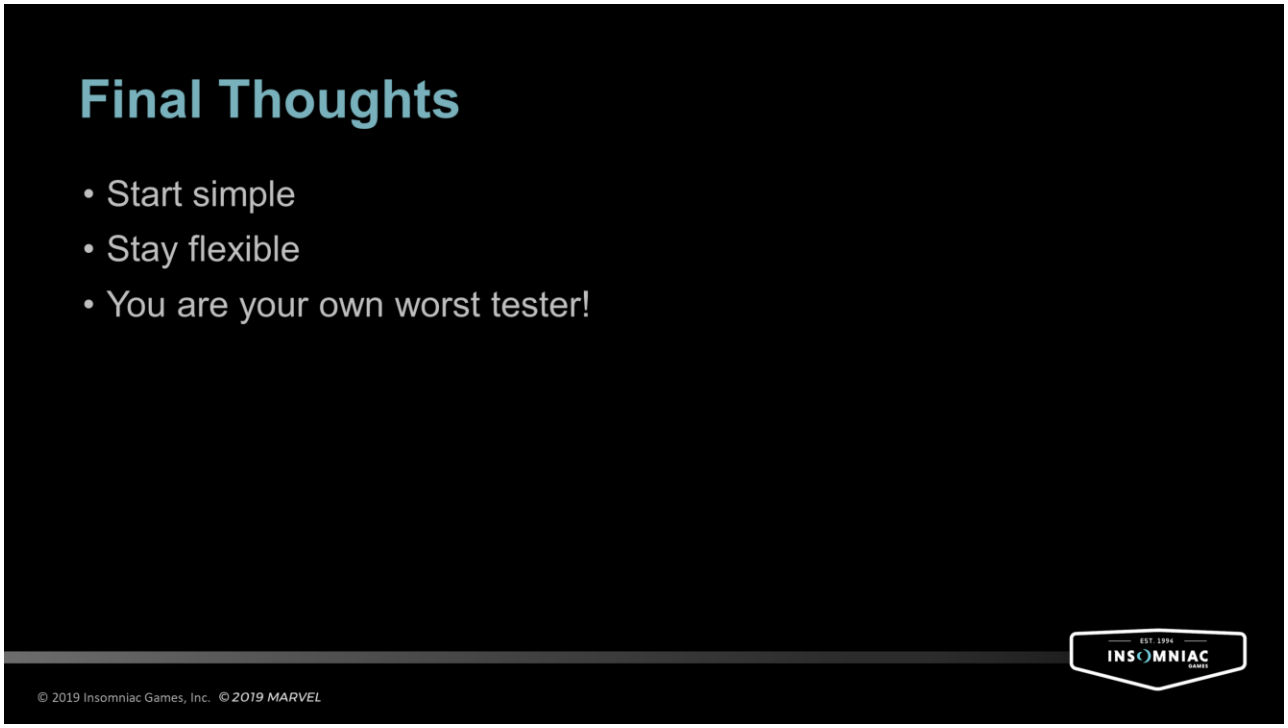
- We ended up with a system with a lot of features but many of the best additions were things we could have never anticipated needing or wanting ahead of time.



- Second, you have to stay flexible in your approach to creating mechanics.
  - We were initially resistant to some of the best changes because we had all the design reasons listed out why it was working like it was supposed to, clearly it was the feedback that

was wrong!

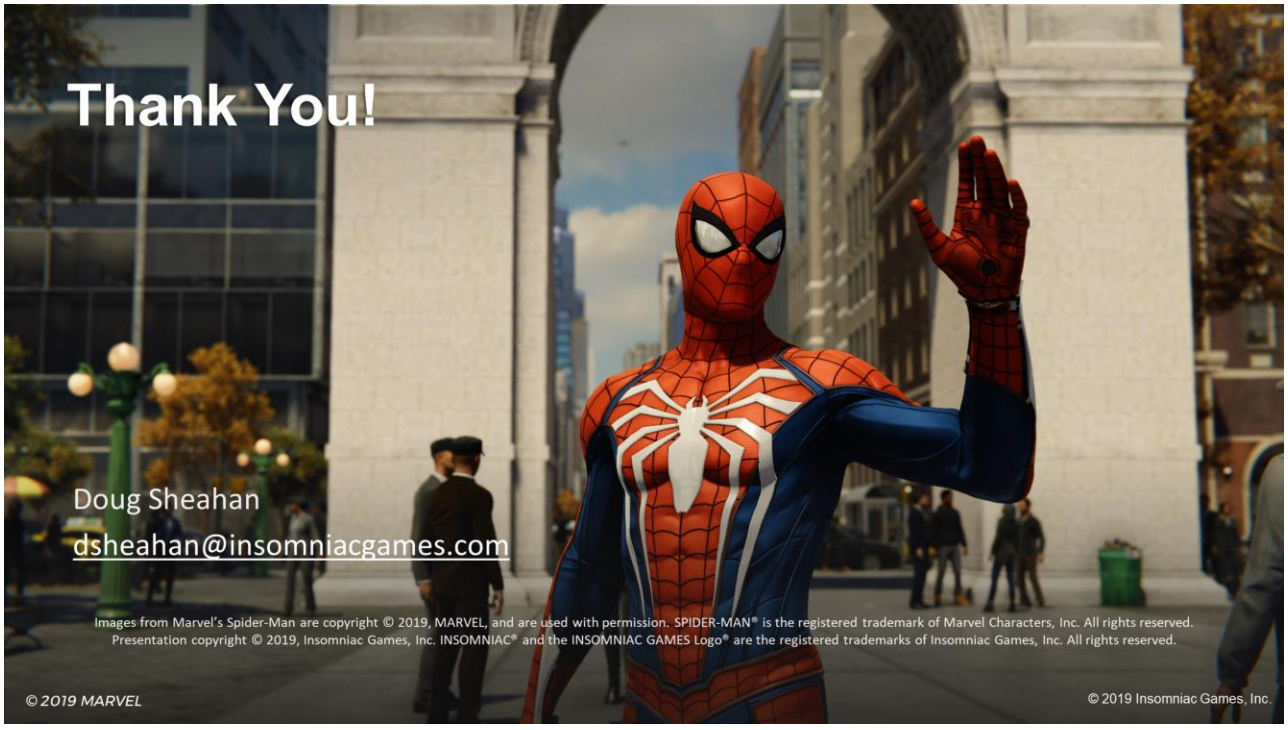
- By allowing the game to evolve we came to a final product that was far superior to what we thought was the right approach earlier in development.



- And finally, you are your own worst tester!
  - Developers know exactly how their systems work and the way they're supposed to be used so when you play the game it will often work exactly as expected.
  - Different players will play the

game in wildly different ways and you will never know where the real problems and edge cases are until you let them loose on your game.

- Even without external playtesters, you should always challenge yourself and others to try and play the game in different ways.



- And that's it for me. Thanks so much for coming.