RIDE'M DOCUMENTATION

Josie Chiao (jyc7ne) - Team Leader | Jonathan Bentley (jbb4ac) | Katherine Qian (kq3zm)
Github: https://github.com/UVA-CS4730-F17/game-project-ride-m

Game Pitch

RIDE'M is a rhythm + runner game inspired by electronic dance music (EDM) culture. Think Temple Run or Subway Surfers—but add a beat. RIDE'M is a combo rhythm-runner game for mobile platforms, where players are an aspiring DJ, playing "shows" by swiping left, right, up, and down and riding to the rhythm, collecting beats and avoiding off beats.

How to Play

One technical thing to note about RIDE'M is that we wrote and built it for iOS exclusively, so be sure to build and play the game via an iOS device.

Gameplay for RIDE'M is simple. When first opening the app, players arrive at a start scene and have the choice of three venues: "Local Party" (Easy) / "Town Bar" (Medium) / "City Club" (Hard). Upon choosing one, they will be taken to play the song assigned to that venue.

Once a song starts playing, beats and obstacles ("off-beats") will start cascading down a note track. To properly play the song, players should swipe their player (which is intended to simulate an equalizer slider on a mix table) left and right to ensure that it's in the right lane to properly catch the notes and play the song. At the same time, there will sometimes be obstacles generated that the player is supposed to avoid. Players can swipe up to avoid those obstacles as opposed to swiping to avoid.

The timing of the beats match up with the audio, so the user is highly encouraged to play the game with the sound on. The frequency of beats correlates with the difficulty of the song—in other words, the more difficult the venue, the more notes there are that the user must catch.

As they play, the player will notice that there are several things going on: most obviously, that as they collect beats, their points (or rep) increase. Additionally, if they continue to collect beats without missing any (or running into obstacles), there is a

multiplier in the top right corner that is increasing in powers of 2. Only collecting beats increases rep—avoiding obstacles does not increase rep. Missing notes and hitting obstacles does not decrement rep, but it does reset the multiplier and also affect “crowd-hype” (i.e. health).

In addition to the multiplier is also “crowd-hype”—”crowd-hype” is the equivalent of health, but it is represented uniquely in the background of the game (as opposed to a traditional health bar). The better players are doing, the more there are people in the crowd, and vice versa: the worse they are doing, the fewer people there are in the crowd. If a player’s crowd-hype completely drops, they’re “boo-ed” off stage and it’s show over. They have the option to try again, or return to Venue Select.

If players reach the highest multiplier (16x), every beat they collect at 16x increments a CO2 meter on the right hand side. Once it is full it will show that it is activated. If the player presses the “Activate CO2!” button that appears in place of the meter, they kick off “Ultimate Crowd Hype”, and their multiplier is increased to 32x for 5 seconds. Additionally, their crowd hype is maxed out. This is essentially 5 seconds of invincibility—if a player collides with an obstacle or misses a note, their multiplier will not be decremented nor will they lose crowd hype, and will stay at 32x. However, they will not accumulate rep for missed notes. After the 5 seconds are over, the player starts off the rest of the song with a 16x multiplier and full crowd hype.

If the player successfully plays the song until completion, then they successfully complete the show and are shown a summary of their results.

Content Available

Currently, the features outlined earlier in “How to Play” are all of the major features. The Song Playing Mode features the core mechanic, swiping to catch beats and avoid obstacles, as well as a point (rep) counter, multiplier, and “CO2 Meter”.

There are three “levels”, which do not differ in features apart from having different audio sources and beat frequencies. These beatmaps were generated initially using an audio processor that analyzed the audio files, generating a pattern for when it detects a beat. We then took this pattern that was originally generated, and used it to generate our beats accordingly.

Playtesting

At the in-lab playtest, we had the core mechanics, a score counter, a multiplier, and a rudimentary game start/game over implemented. Playtesters responded very positively to the graphics and audio experience, but had comments concerning feedback. Many of them suggested that we provide more feedback for when a beat was successfully caught, or for when they ran into an obstacle. A few noticed and commented on how our notes weren't exactly lined up with the audio just yet. From this playtest, we got to hear several good ideas for how we could improve the feedback, and also were able to get a good idea of which features we really needed to focus our energy in on first.

In our second iteration of playtesting, we had three non-CS 4730 students with close to no background understanding of RIDE'M playtest our game. Between the in-lab playtesting and this playtest, the primary feature we'd added was the Crowd Hype indicating background, since several playtesters from our class remarked that there needed to be more feedback (i.e. a health bar) to indicate how they were doing, and why the game ended. At the time we were still working on improving the beat-audio synchronization (which took a lot more time than anticipated), so we didn't have that quite ready yet.

For these playtesters, we provided the device and told them they were playing a music game "like Guitar Hero", and gave them the start screen. Most of our playtesters picked up the game mechanics very quickly, but sometimes needed to have the mechanics clarified to them. For example, one playtester didn't realize they were supposed to swipe, and instead attempted to tap at first. One playtester also didn't realize that they were supposed to avoid the obstacles, but learned it quickly after seeing that it reset the multiplier.

The difference between this group of playtesters and the previous could be their familiarity with our project—while most in our lab probably recalled that RIDE'M was intentionally *not* a traditional tap-to-play rhythm game, these playtesters didn't have that background information.

But similarly to the previous group, they responded positively to the look and feel of the game, and felt that the unique representation of health (the crowd-hype background) was also cool—there were no longer any comments about being unaware of how they were doing while playing through the song. They did, however, also note that they

noticed the beats could've matched up with the audio better—to which we responded that we were already working on improving that part of it.

Since playtesting, in this final iteration before the Expo, we've added various ways to accommodate for the lack of feedback. We now have the player glow upon catching a note, and various sound effects as well.

Lessons Learned

It's hard to effectively divide and conquer development.

One of the largest things our team struggled with was git and version control. It wasn't the tool itself, rather, its integration with Unity that proved to be much more difficult than we'd hoped, even with a properly configured .gitignore. Essentially, we found that it was more difficult than we had expected to break up work in game development in a way that could have us working effectively and efficiently in parallel. Even though we thought carefully about how we could split up features in a way that would have us working on reasonably different parts of the game at all times, because we were usually all working on the main scene, we often ran into trouble with integrating all of our changes together (since there were always merge conflicts, and often in binary files that we didn't understand).

It is easy to accidentally lose sight of aesthetics when you start getting caught up with your mechanics.

The original game design document for RIDE'M outlined and emphasized a strong narrative point to the game. But as we began development, we prioritized laying down core mechanics and from there, continued to add functionality—and the narrative piece started to take more of a back seat. In the interest of time, we chose to prioritize the former; naturally, some features took longer than others and some took a lot of time to not only implement, but also *improve*. We certainly learned why in game development, this work is often split among different teams of people—storywriters focus on that side of the game, and developers focus on the functionality. There is always crossover, but there are so many parts of a game, and a few members can only effectively focus on so much.

At the same time, multifaceted thinking and skills are valuable.

While we primarily put on the developer hats in the interest of time and in the context of this class, each of us contributed to the experience of the game in different ways, attesting to the importance of having multi-faceted members in a game development team. As discussed in class, games are a very unique category of the tech industry—it is beneficial for each and every team member, regardless of what their specific role is, to have a sensitivity for the other parts of the game. For example, one of our team members had experience with 3D modeling that was invaluable to the polish of our game. Another had experience with graphics, and were thus able to create sprites, splash pages, and backgrounds for RIDE'M. Another had experience editing audio, so they were able to clip and make minor changes to sound clips we downloaded to match up with what we wanted. As a team of computer science students, we all met each other's ability to develop features and code mechanics—but the ways in which we individually contributed with our other experiences were honestly just as important.