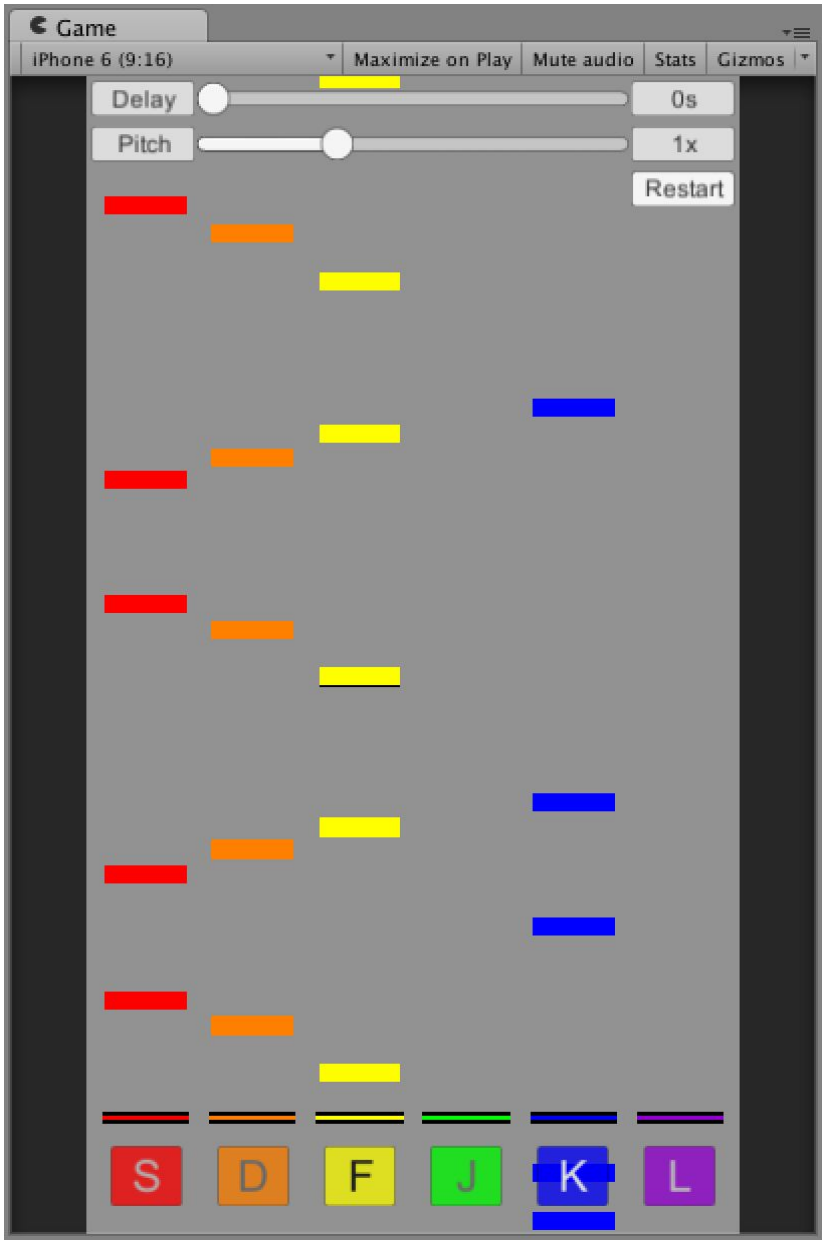
Koreographer Rhythm Game Demo Overview

for v1.5.1

Table of Contents

Overview

The Rhythm Game Demo content for Koreographer is very simple. It shows how you can use Koreographer to create traditional rhythm game gameplay. The demo content was developed in Unity 5.0 and should be forward compatible with newer Unity versions.



The Rhythm Game system uses a combination of Unity UI buttons and Unity Sprites for visuals. The demo scene was developed with a target Aspect Ratio of 9:16. Please be sure to set your Game View *Aspect* to a ratio of 9:16 for best visual results. The aspect ratio was chosen to match a common mobile device aspect ratio.

Koreography is used in the Rhythm Game Demo to drive the gameplay. Each *KoreographyEvent* in the *KoreographyTrack* with the "Piano" Event ID will drive an individual Note Object. Each *KoreographyEvent* uses a *TextPayload* to specify which vertical Lane it should appear within.

The Timing Targets are positioned right above the onscreen buttons. The precise position that represents "Now" is the center colored line. The Note Objects themselves are vertically resized based on how much of a timing window is configured. The goal of the game is simple: touch the correct button while a Note Object is beneath the Timing Target visual.

Several configuration buttons appear at the top of the screen. *Delay* is used to assist in timing issues that appear on certain platforms due to audio processing latency. Mobile platforms are especially susceptible to this issue. Tests with an iPhone 6 and default Audio Latency settings show that a Delay value of ~0.1s (100ms) seems to work well. *Pitch* will change both the pitch and playback rate of the audio. Changing this value will change the playback speed and, thus, the rate at which Note Objects fall towards the Timing Targets. Clicking or pressing the *Restart* button will restart the demo song.

See the following sections for more detail.

Rhythm Game Assets Overview

The Rhythm Game Demo contains the following assets.

- **Scripts**
  - System Scripts
    - **RhythmGameController** - Controls the high level rhythm game logic. Universal gameplay values are configured on this component, including Note Speed, *LaneController* specification, Lead In Time, and more. The Event ID of the *KoreographyTrack* to source for Note Objects is also specified here.
    - **LaneController** - Manages the logic for an individual Note Lane. Lane-specific values are configured on this component, including Color, Target Visuals, the Keyboard *KeyCode* to use for input, and the *TextPayload*s to match *KoreographyEvent*s against.
    - **NoteObject** - Controls the logic governing the position of a single Note Object while the game plays. It interacts with both the *LaneController* and the *RhythmGameController* to properly configure its position while active in the scene.
  - Support Scripts
    - **ButtonController** - Drives button related setup. Sets the color of the buttons and, when on non-mobile platforms, initializes the *Text* component with the specified Keyboard Keys that control button input.
    - **DemoDelayDisplayUI** - Handles Delay UI changes.
    - **DemoPitchDisplayUI** - Handles Pitch UI changes.
- **Textures**
  - **UIElements** - A simple texture atlas that contains graphics used by the *SpriteRenderer* and UI *Image* components in the game.
- **Koreography**
  - **RhythmGameKoreo** - The *Koreography* data. This links the *AudioClip* to the *KoreographyTrack*s.
  - **PianoTrack** - This *KoreographyTrack* is configured with the Event ID “Piano”. It specifies the timing of all notes that make up the rhythm gameplay. *KoreographyEvent*s in this track are OneOffs with *TextPayload*s. Each *TextPayload* contains the note of the piano key hit at that position in the demo song. Text values are the same as those output by the Pro MIDI Converter.

Rhythm Game Logic Flow

Main Game Flow

Logic in the Rhythm Game Demo is very simple. At a high level, the Rhythm Game Controller accesses the *KoreographyTrack* that contains all of the music’s game events. Each *KoreographyEvent* in this track contains a *TextPayload* that determines which of the six lanes it should appear within. When the game begins, the Rhythm Game Controller accesses the *KoreographyEvent*s in the *KoreographyTrack* and filters them, passing them to the matching Lane Controller. As the music plays back, each Lane Controller checks to see if it should spawn any Note Objects for *KoreographyEvent*s that will soon be visible. It performs this operation by looking at the Timing Target location, which defines the position at the *current* music time, and compares it

with the spawn location (just above the top of the screen). The “music time at spawn location”, which will be sometime *after* the current music time, is calculated to take the configured speed of the notes into account.

Each Lane Controller is configured with both a UI button, as well as a Keyboard *KeyCode*. When the associated button or key is pressed, the Lane checks to see if any tracked Note Objects happen to be at the Timing Target position. If so, the Note Objects themselves are immediately removed from the screen and the assets returned to a pool controlled by the Rhythm Game Controller. If a Note Object is not hit and moves out of range of the Timing Target, the Lane Controller stops tracking it.

Note Object movement is controlled by the Note Objects themselves. When updating their position, they take into account certain settings located in both the Rhythm Game Controller and the Lane Controller. When the Note Object moves beyond the calculated despawn location (just below the bottom of the screen), it deactivates itself and returns its assets to the pool managed by the Rhythm Game Controller.

Lead In Time Handling

Many rhythm games require a “pre-roll” or “lead-in” phase to show Note Objects before music playback actually begins. The Rhythm Game Demo includes a working example of how this might be implemented. The included example provides a mechanism by which the current time can be queried as *DelayedSampleTime* in the *RhythmGameController* class. During a lead-in phase, the value returned by *DelayedSampleTime* is negative. As Note Objects position themselves based on *offsets* from this time, they are able to properly position themselves with the same logic used when the *DelayedSampleTime* is positive. See the `NoteObject.UpdatePosition` method for the specific logic.

Audio Latency (Delay) Handling

Many platforms (especially mobile devices) suffer from a relatively constant amount of Audio Latency. This latency can depend on both hardware and OS version. As such, it is often best to provide the player with an opportunity to adjust the timing system to suit their specific setup. The Rhythm Game Demo makes use of Koreographer’s Event Delay system to account for this. A UI *Slider* positioned at the top of the screen allows the player to adjust Koreographer’s Event Delay setting while the game runs. The *Restart* button does not reset this value.

Other Sources of Delay

It should be noted that there are two other major sources of delay that *could* impact your game’s feel:

1. Display Lag - The amount of time between:
    a. The time at which the display receives the frame it should display.
    b. The time at which the display actually displays the frame.
2. Input Lag - The amount of time between:
    a. The time at which a player physically inputs a command (keyboard, mouse, controller, etc.).
    b. The earliest time at which the game can process the input.

As with Audio Latency, these tend to be device or hardware specific and reasonably constant. The Rhythm Game Demo does not currently provide an example of how to specifically mitigate these lag sources.