Dasher Game Mode

Description

Game mode is a module written for Dasher whose intention is to provide new users with a way to learn how to effectively use the application. Dasher itself can be difficult to understand at first glance. It can be even more difficult to internalize the mechanics behind movement through words. Game Mode, at its most basic level, provides visual guides for entering a target string of text. It is modeled after traditional typing tutorials that give you a passage to type and helpful responses if you type the letters correctly or make a mistake. Please note that this has only been tested on Gtk systems, so far.

Architecture

Game Mode has been designed to be as modular and flexible as possible from the ground-up. It also has been designed to require as little change on the part of the core code-base as possible, keeping in line with the plug-in mentality. In order to accurately tag nodes, there *have* been a few parts of core code that have been modified to support game mode. Most notably, Dasher Model now handles searching through the tree for nodes as it is the only structure with that information in the first place. DasherInterfaceBase also had to be augmented in order to integrate game mode in.

The core class at work here is the *GameModule* class. This is a class that inherits from *DasherModule*, which means it is a *DasherComponent*. Being a Component, it takes advantage of the completely generic event system to interact with Dasher. It subscribes to and responds to Text Draw events and Edit events. It uses the Edit events to know whether a user has typed the correct character or not and respond accordingly. It uses the Text Draw events to acquire the position of the labels on nodes to draw a custom crosshair over the next letter a user must type.

To determine what words a user types, Game Module employs the help of a Word Generator. Word generators define an interface that allows clients to read words from a source defined by the concrete implementation. For example, there exists a *FileWordGenerator* class that inherits from *WordGenerator*. Its purpose is to generate words by simply reading from a file. Game Module calls *NextWord()* to get the next word from the generator.

If a game text file in not specified by the user when starting game mode, we use the current alphabet's default file. The name of a default game file should be game_mode_ *alphabet-name*.txt. For example, the game text for all English alphabets models is game_mode_english.txt. These files need to be placed in Data/gamemode to be installed in the proper location. To make an alphabet "aware" of its game text, add a <gamemode> tag to its XML definition, containing the name of the game mode file. See alphabet.english.xml for an example.

Game Module defines the block of words that are displayed as targets for the user as "chunks". The size of these chunks, in # of words, is defined as a constant member. So, an example use case for Game Module would be the generation of a chunk. If the chunk size is three, *FileWordGenerator::NextWord()* would be called three times, appending the results to the member of Game Module that holds the target chunk, *m_sTargetString.* The benefit here is that Game Module no longer cares how the words are obtained. Any implementation of the *WordGenerator* interface can be used without issue.

In terms of how Game Module actually draws to the canvas, this should be kept platform independent and should make use of the Dasher drawing functions that have been implemented on all platforms. The drawing happens in *DecorateView()*. We chose this name because it's very descriptive, but it's important to realize that though Game Module and *InputFilter*s share a method in name, they are NOT at all equivalent. Game Mode does not (and should not) modify the way a user interacts with Dasher in terms of input.

In order to display the chunks that the user must type, Game Module makes use of the GameDisplay class. GameDisplay is a platform agnostic abstraction layer that allows Game Module to draw colored text to a region on Dasher's main window. In order to compile, GameDisplay must be subclassed by a platform specific implementation. For example, in Gtk, GtkGameDisplay is responsible for this. GtkGameDisplay uses the Pango text markup language to display chunks with color highlighting. Though GameDisplay is currently responsible only for displaying chunk text, it can easily be extended to display more information, eg. a score, time left, etc. To do so, simply define the new UI elements, define new virtual methods in GameDisplay, and implement those methods in a platform specific class.

Initialization

GameModule is constructed as a module in DasherInterfaceBase, at which point it is immediately registered with the module manager. When DasherInterfaceBase::InitGameModule() is called, a pointer to the Game Module is fetched from the module manager and assigned to m_pGameModule. The game module is then given a word generator that, depending on whether SP_GAME_TEXT_FILE is set, uses either the current alphabet's default game text file, or a user specified text file.

Because GameDisplay is a purely abstract class, m_pGameDisplay must be constructed in the platform specific extension of DasherInterfaceBase. Once the GameDisplay is instantiated, a pointer to it is given to the Game Module, so that it can begin drawing.

When game is terminated for any reason, m_pGameModule is set to NULL. The actual Game Module and its Game Display, however, are not destroyed because of this. The Game Module still exists, fully constructed, in the module manager. When game mode is started again, a new pointer to that same Game Module is simply fetched from the module manager.

Usage

To use game mode, navigate to the file menu and click "Game Mode". This displays a dialog box prompting the user with two options – choose a text file to play game mode, or choose the default one for the current language. When one of these options is selected, BP_GAME_MODE is set to true, and SP_GAME_TEXT_FILE is set to the appropriate file path. The game display then unfolds from under the Dasher editor to display the first chunk from the text file.

Issues

Currently, there are a few issues to work on:

- If a letter one must type does not appear on screen, the crosshair should disappear. Currently it remains visible, but does not move.
- There is currently no response for not typing the correct letter.