Talent Minigame Development Guide

Table of Contents

1 Overview

Warning: The talent minigame API is still in development and is subject to interface and implementation changes.

This guide presents information on the design, art and engineering processes required to develop talent minigames for Pixie Hollow. Third-party developers will create additional minigames using the Pixie Hollow minigame SDK (*see below*). To ensure consistency with the rest of the MMO, Disney Online, Schell Games and SilverTree Media will review each game through its development process.

2 Talent Minigames

Fairies will encounter a number of talent-themed minigames in Pixie Hollow. Hosted by famous Fairies, these games will introduce players to many of the activities that the different Fairy talents perform. They will also reward player Fairies with ingredients, effectively providing another form of gathering.

2.1 Themes

Each minigame is associated with a particular talent and presents an activity or duty that Fairies of that talent perform. These games are an important way to reinfornce the Fairies' involvement with nature and the seasons. To that end, they also involve animals as much as possible. The "Pixie Preview" animations - appearing on the Disney Channel and elsewhere - are great examples of talent activities and animal friends.

In the Fairies universe, each Fairy has special skills that are specific to her talent. In the upcoming "Tinker Bell" movie, Tink isn't able to interact with water or light in ways that Silvermist and Iridessa can. Because of these skills (and since they love what they do!), Fairies stick to their talent tasks year-round.

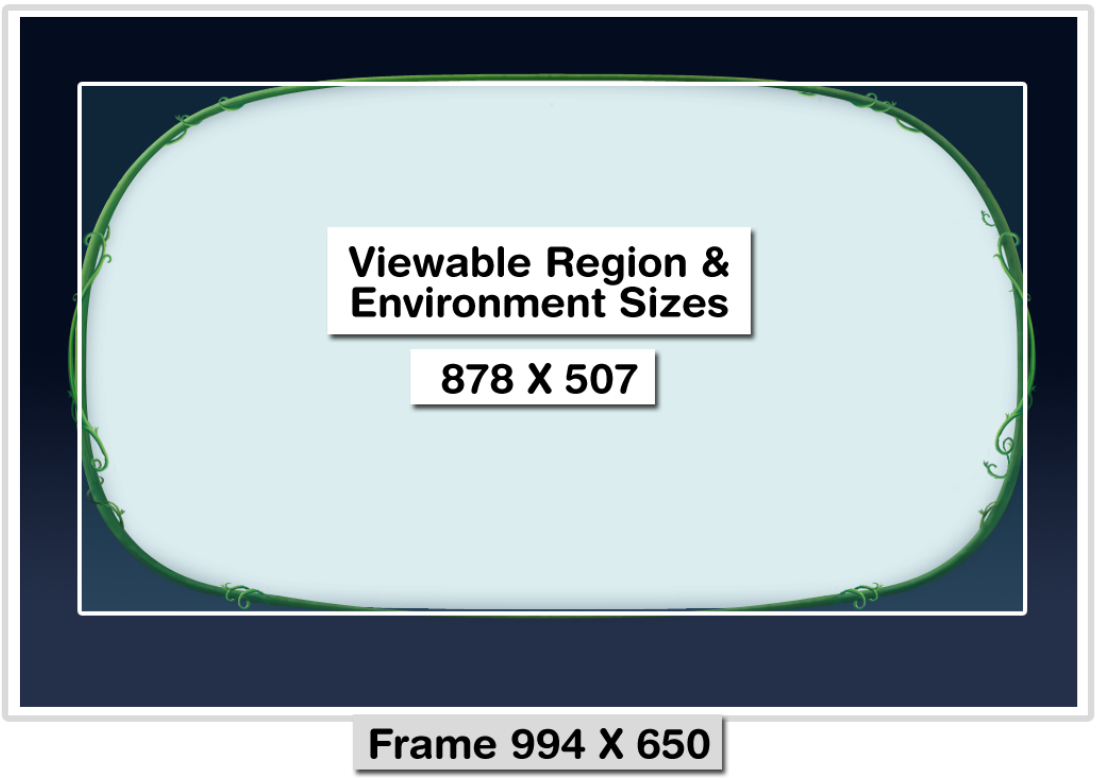
However, it's important that players have access to all talent minigames - regardless of their Fairies' talents. Therefore, these games present opportunities to help Fairies of the associated talents rather than perform their tasks directly. The player's Fairy will work with a famous "host" (*e.g.* Silvermist) in each game.

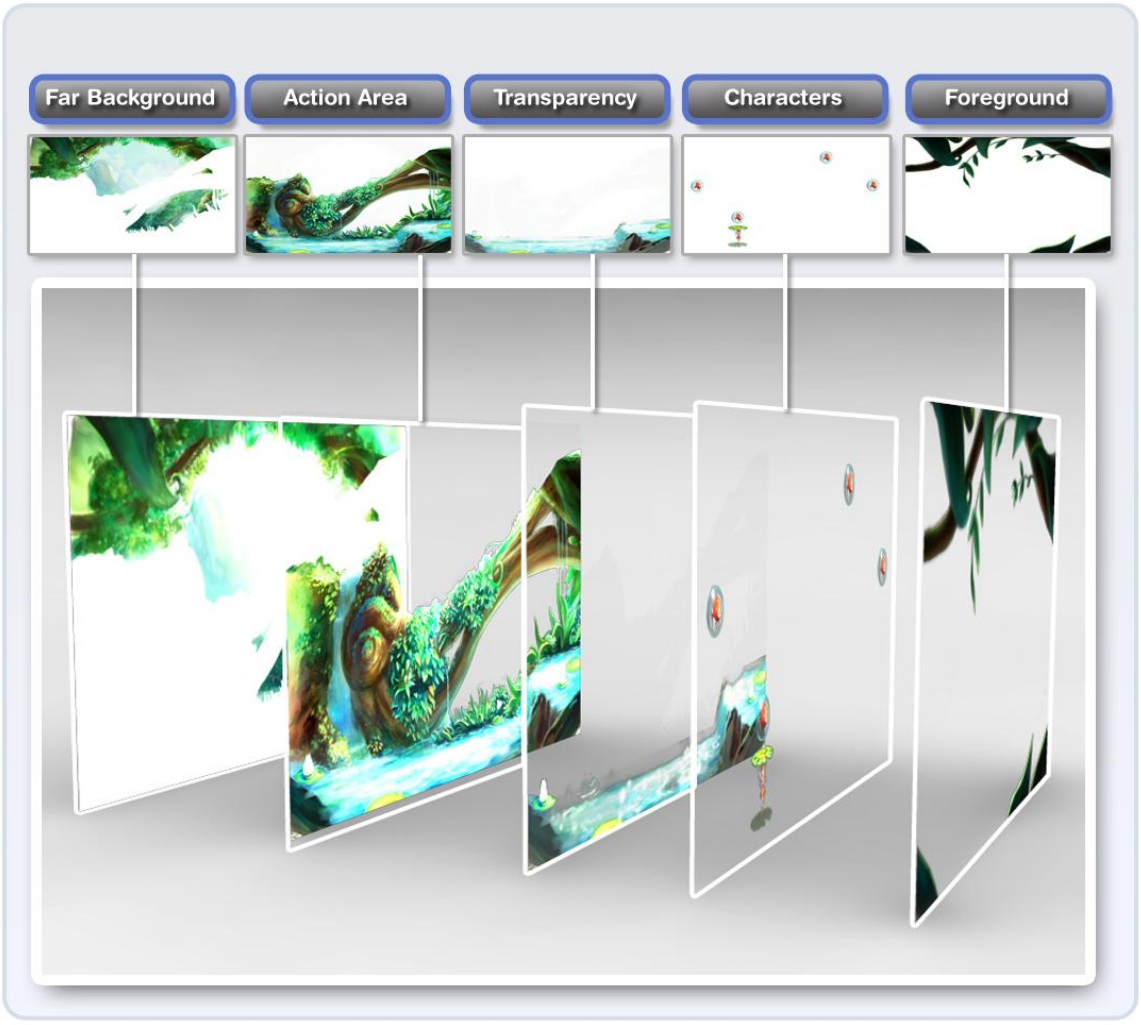
2.2 Game Meadows

Fairies will access talent minigames through themed gates that appear in meadows around Pixie Hollow. Like other meadow gates, they transport Fairies to new locations - this time to special "minigame meadows" for the talent-themed activities. These separate meadows not only help ground the activities in Pixie Hollow, giving them a nice backdrop, but also provide a number of technical benefits. For example, the Flash client can focus on the minigame and avoid rendering other Fairies that are entering and exiting, chatting, etc.

3 Environments

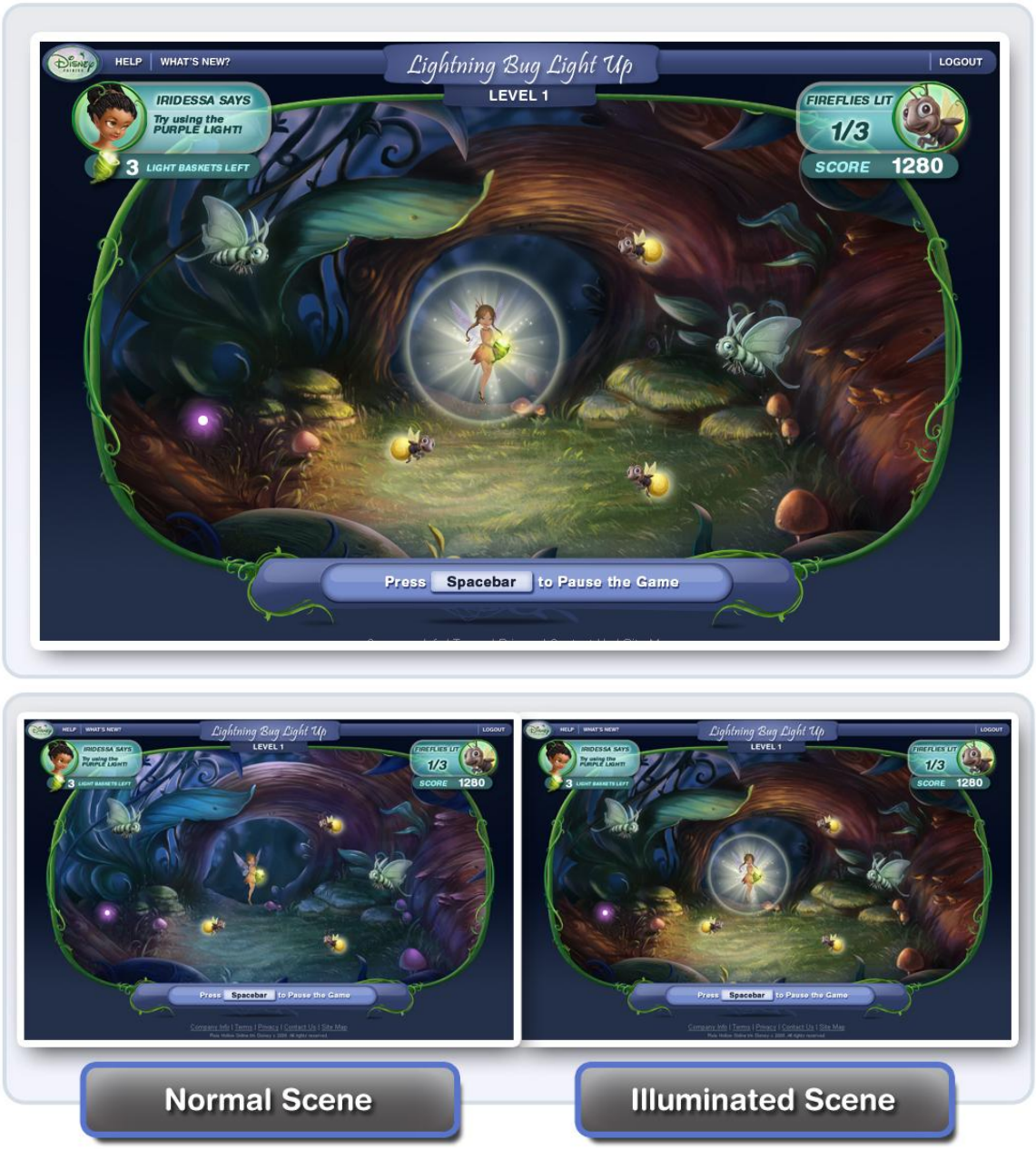
Like normal meadows, minigame environments will consist of several layers. For performance reasons, however, games should constrain their activities to one screen and avoid multiplane movement. Multiple layers for minigame meadows do allow the game to support environmental effects such as cloud movement or shimmering rays of light. Foreground planes can also partially obscure characters, making them feel grounded in the environment.





Separate layers for the "Bubble Bouncer" minigame

The "Firefly Light Up" minigame (light-talent) uses two sets of layers – normal and illuminated. During the game, the layers can blend to produce a light up effect.



"Firefly Light Up" blends two layers to create a light up effect

4 Characters and Items

4.1 Player Fairy

In these minigam e meadows, Fairies don't appear and fly around as usual. Instead, each game defines a unique appearance and control scheme for the player's Fairy. For example, the Fairy holds a lily pad above her head in the "Bubble Bouncer" game and carries around a basket in "Firefly Light Up."

4.2 Fairy Host

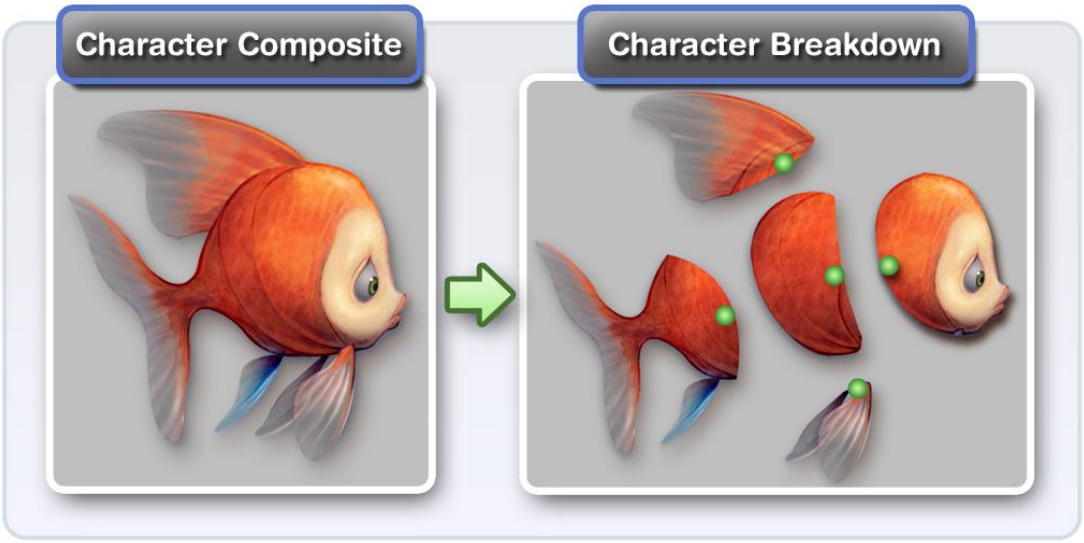
Each game features a famous Fairy with the associated talent. These hosts guide players through the experience, offering help, guidance, and encouragement. They are also an important way to reinforce the talent theme and the overall Fairies universe.

In the initial minigames, the Fairy host appears in the upper-left corner of the game screen. During the game, her image changes to a different expression as appropriate. To match the rest of the game, these images are based on Disney Toon Studio's "vis dev" illustrations.

Joyful, relaxed and concerned states for Silvermist and Iridessa

4.3 Animals

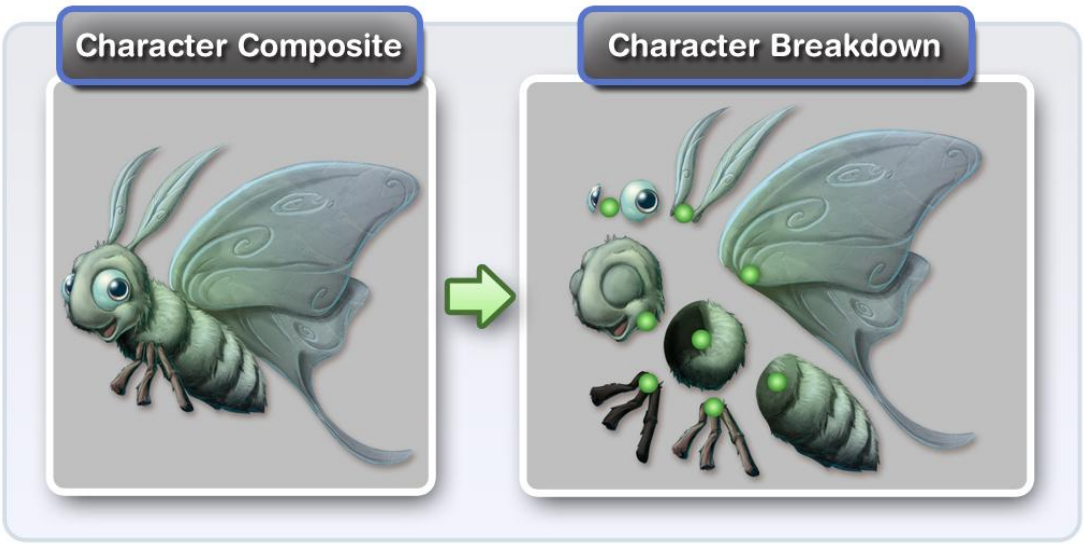
As mentioned above, talent minigames involve animals as much as possible. These animals consist of many pieces to facilitate their animations.



Green dots represent the pivot points of each piece

Characters are broken down for several reasons:

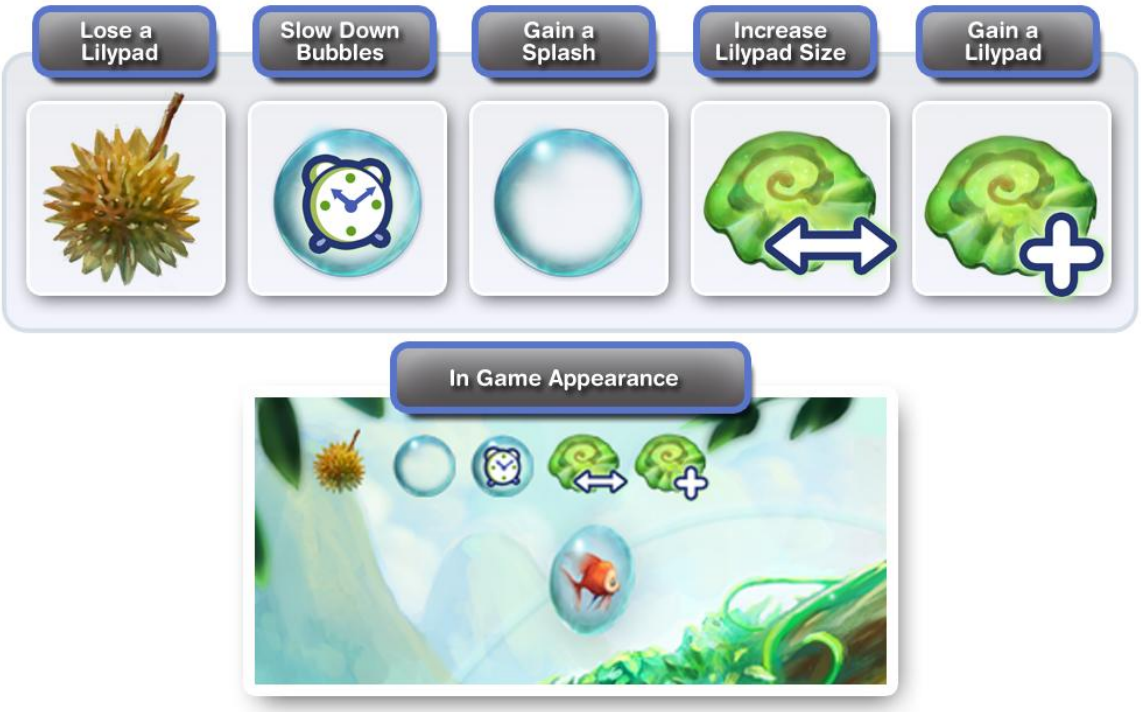
- File Size
  - To limit the amount of individual drawings that Flash loads, all characters rely on a simple skeletal system. Individual pieces are animated, creating a sense of movement.
- Ease of Animation
  - Flash can easily animate characters with shape and motion tweening. Characters should always use this method over hand-drawn keyframe animation.



Animal characters for "Firefly Light Up"

4.4 Game Items

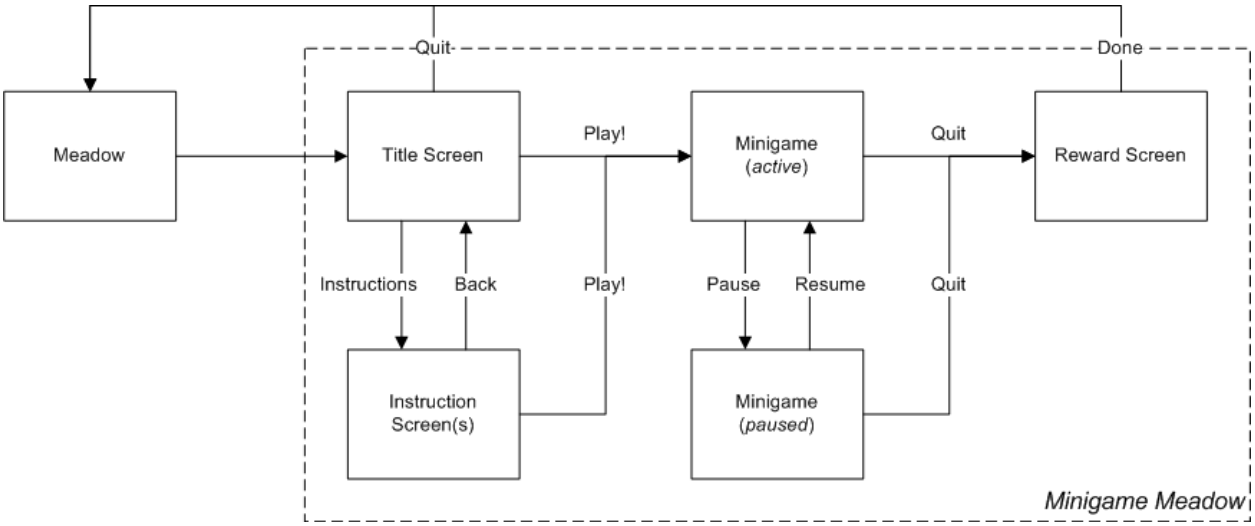
Below is a sample set of in-game bonus icons. These icons should be cartoonish, with a touch of realism, for readability. As always, avoid using text in these images due to localization concerns.



In progress – replace clock with hourglass and add '+' to splash/miss icon.

5 Interfaces

To present a consistent experience for players, talent minigames use a standardized framework. This framework supports the following flow:



Needs a "replay" arrow from reward screen to minigame

The minigame meadows also make the following adjustments to the Pixie Hollow interface:

- Leaf Jounal
  - *Not available* – prevents player from teleporting away prematurely
- Profiles
  - Available, but "Fly to" and "Visit home" are disabled to prevent the player from teleporting away prematurely
  - If a friend tries to "fly to" the player, she receives a "your friend is busy" message instead
- Friends list
  - Not available while the minigame is active (*see "Playing and Pausing" below*)
- Chat
  - *No broadcast chat* – the games are single-player experiences, so the player is effectively alone in these meadows
  - Not available while the minigame is active (*see "Playing and Pausing" below*)
- Alerts
  - Appear as usual, but clicking certain alerts (*e.g.* earned badges) will not bring up the Journal
- Whisper bubbles
  - Appear as usual, but clicking these while the minigame is active will pause it (*see below*)

5.1 Meadow Signs

Special signs appearing in Pixie Hollow's meadows provide access to the talent minigames. These signs only use graphics – the name of the game will appear on a tooltip.

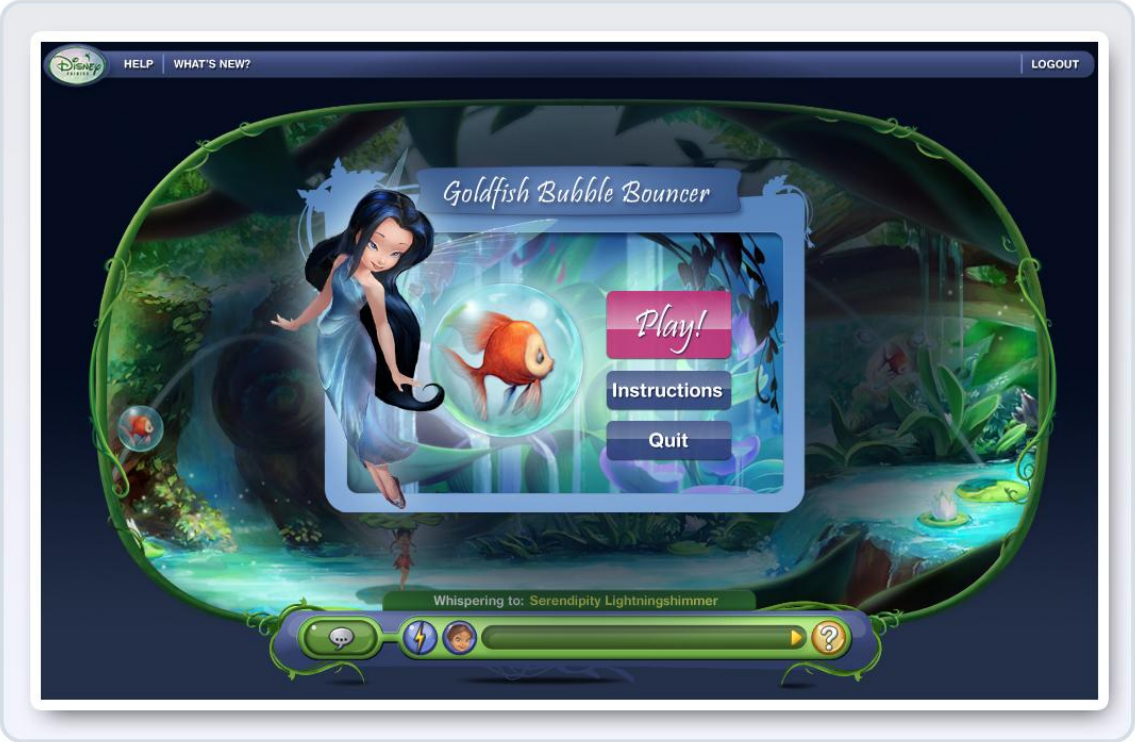
Signs use iconic graphics for several reasons:

- Clean Presentation
  - The overall appearance of Pixie Hollow should be clean and clutter free. Graphics appear instead of text wherever possible.
- Fun for the target audience
  - It's more fun for players to see an engaging image of what they are about to do rather than read about it.
- Seamless Localization
  - All text in Pixie Hollow must be externalized to support localization. Graphics should never contain baked in text.

5.2 Title & Instructions Screens

As soon as a player arrives in the minigame meadow, the game's title screen appears. On this screen, the following options appear:

- Play!
  - Activates the minigame (*see below*)
- Instructions
  - Displays the game's control scheme and goals
- Quit
  - Transports the player back to the previous meadow



"Bubble Bouncer" title screen

If the player navigates to the instructions screen, she has the following options:

- Play!
  - Activates the minigame (*see below*)
- Back
  - Returns to the title screen
- Next/Previous (*optional*)
  - Navigates through multiple instruction pages if applicable

Each minigame supplies its own graphics and text to create a unique design for these screens. Every game must also display the above flow options (keeping the same labels), but the layout and style for the buttons can vary. To avoid any confusion, these buttons should be clearly visible and (when possible) follow a common layout.

On each of these screens, the Friends list and chat bar (whisper mode only) are accessible.

5.3 Game Screens

After the player chooses the "Play!" option, the minigame activity begins. Each game defines its own controls, visual appearance and goals. Pixie Hollow is primarily a mouse-driven experience, though, so most games will focus on mouse controls.

Active game screen for "Bubble Bouncer"

The minigame SDK (*see below*) provides a number of standard heads-up display (HUD) elements. This helps all the talent minigames provide a consistent look and feel.

HUD elements for "Bubble Bouncer"

1. **Fairy Host & Lives Remaining**
   The host Fairy appears in this corner, chatting and occasionally changing her expression based on the game's events. This element also displays the remaining number of lives for the player.

2. **Game Title & Current Level**
   The game's title is visible on all screens. During gameplay, the current level appears below it.

3. **Game-specific Progress (*optional*)**
   The appearance and function for this optional element depends on the minigame. In this example, the meter displays the number of fish bubbles the player is allowed to miss in "Bubble Bouncer."

4. **Goal & Scoring Information**
   This element reveals the player's progress towards the level/game goal and her current score.

   Note: In the above example, the score multiplier meter – displaying "x2" – is game-specific and not part of the standard interface.

5. **Chat Bar (*disabled*)**
   A minigame disables the chat bar and friends list while it's active. This prevents the player from disrupting her game until she decides to pause (*see below*). It also allows the use of keyboard controls.

5.3.1 Playing and Pausing

It's important that players have the ability to whisper to their friends while in these minigame meadows. Most talent activities are skill-based, however, and stopping to chat will likely disrupt the player and damager her progress/score. Some games may also use keyboard controls, creating a conflict with the chat bar. To solve these issues, minigames will offer two states - *active* and *paused*.

Active

While the minigame is active, all chat options and access to the Friends list are unavailable. Instead of Speedchat buttons and a text entry field, the chat bar interface displays instructions for pausing the game. It instructs the player to hit the *spacebar* to pause the game.

Since most games will provide mouse controls, hitting the spacebar (the largest button on the keyboard) is least likely to disrupt the player mid-game. For players who really want to stick to the mouse, however, clicking the message on the disabled chat bar will also pause the game (*TBD*).

Paused

When the player pauses the game, the action freezes and the game's instructions reappear in the diecut. This screen provides similar options as before:

- Resume
  - Returns the game to the active state
- Quit
  - Takes players to the rewards screen (*see below*)
- Next/Previous (*optional*)
  - Navigates through multiple instruction pages if applicable

In addition, the chat bar is now operational in whisper mode (no broadcast option in minigame meadows). The Friends list is also available in its usual position in the upper-right corner. This allows players to chat without negatively affecting their game. Since the chat bar is now consuming keyboard input, the player will use the mouse (by clicking "Resume") to return to the game.

5.4 Transition Screens

Each game is different, defining its own goals and pacing. While these aren't strictly required, the following mechanics are strongly recommended:

- "Next Level" screen
  - Separating a game into levels lets the player take occasional breathers – and allows the game to *congratulate her* on her progress!
  - *Helpful hints* and themed dialog can appear, possibly based on the players progress.
  - These screens also provide a way for the player to *quit* without interrupting gameplay.
- "You Win"/"Game Over" screens
  - Almost every talent minigame will present a skill-based activity, including a limited number of lives or turns.
  - When the player has completed all levels or lost all her lives, this screen transitions her to the *rewards* screen.

The minigame SDK also provides these transition dialogs.

TBD – "Next level" and "game over" mockups

5.5 Rewards Screen

Talent minigames reward players with ingredients. This effectively makes these games an alternative form of gathering (a faster option for skilled players). Like the gathering system, the ingredient rewards will change throughout the year to reinforce the seasons and tie in with various community events. If a player participates in a minigame and earns any kind of score, she receives a reward.

When the player is ready to move on to other activities, she can choose the "Quit" option in one of the following places:

- Pause screen
- "Next level" screen (*defined by the minigame*)
- "Game over/Try again" screen (*defined by the minigame*)

Next, the reward screen presents the ingredients that she earned based on her game score. Each minigame defines the ingredient(s) that it rewards as well as a point-to-ingredient conversion rate. Like the title and instruction screens, the game can also provide graphics for the rewards screen. However, the actual prize interface – showing the ingredients and the amounts – is part of the standard minigame interface.

On this screen, the Friends list and chat bar (whisper mode only) are accessible. The player also has the following options:

- Replay
  - Restarts the minigame (without returning to the title screen)
- Quit
  - Transports the player back to the previous meadow

TBD – Reward screen mockup

6 Software Development

The talent minigame SDK provides access to a subset of the Flash API and Pixie Hollow framework API. All talent minigames must use this SDK – otherwise, they cannot reliably integrate with the MMO game client.

6.1 Developer Requirements

Talent minigame developers will need the following tools:

- Adobe Flash CS3
- Apache ANT
  - Provided with Eclipse or available at http://ant.apache.org
- Subversion Client
  - Tortoise SVN – http://tortoisesvn.tigris.org

A Subversion repository provides the talent minigame SDK and related documentation. Developers will add and commit their files to this repository. Schell Games and SilverTree Media will also use this repository to update the SDK as minigame requirements change. The repository contains:

- Command-line Flex compiler
- Precompiled unit testing suite
- ANT build scripts
- Example minigame source

Developers are expected to deliver the following components:

- Game design documentation
- ActionScript package
- Content FLA
- XML configuration file

Schell Games and SilverTree Media will review all minigame code before merging it into the MMO game build.

6.2 Getting Started

6.2.1 The Talent Minigame SDK Repository

Checkout the FairiesTalentMiniGameSDK repository. The unit testing suite and example minigame exist inside this folder. To view the example game, launch **/index.html**. This displays a link for the example minigame – select and run it. The unit testing suite uses local dummy data and does not connect to a live game server.

The SDK folder

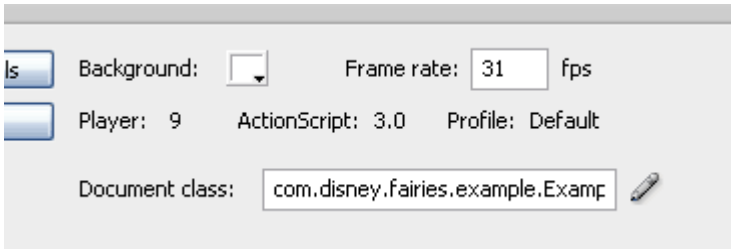
Note: You may need to relax your Flash Player security settings to allow local content. The Flash Player security panel is available online at http://www.macromedia.com/support/documentation/en/flashplayer/help/settings_manager04.html

The Flash Player security panel

6.2.2 The Minigame Content FLA File

When creating a new minigame, developers should copy example game files to ensure that all content requirements are met. To do this, make a copy of **/fla/example.fla** and name it something like **/fla/mygame.fla**. Open this new file in Flash CS3. This file does not contain ActionScript, and it was not published against any external ActionScript.

The Document Class in the properties panel is the next important detail. The example file points to **com.disney.fairies.example.ExampleMiniGame**. This class is not actually published into the content SWF. Flash will generate an empty version of this class, and the framework will swap this file out at runtime. Change this to **com.disney.fairies.mygame.MyGame**.



The Document class property

Gray wireframe versions of all the necessary game content are on the stage. On the top-level are MovieClip symbols for the primary game screens: `title`, `instructions`, `prize`, `game`, and `bg`. The Pixie Hollow framework manages the title, instructions, and prize screens entirely. Developers only need to provide content skins for them in this Flash file. Refer to the content requirements section for details on these content symbols.

The `game` symbol is the root for the actual minigame that developers will contain all the developer's scripts and contents. The `background` symbol represents a common, optional background that the game can shut off during gameplay.

Edit your new file to contain your desired content, and change its publish settings to publish to **/swf/panels/MyGame.swf**. Remember that *absolutely no ActionScript* should be published into this SWF. The framework will not run this SWF directly.

6.2.3 The XML Configuration File

Because the minigame is delivered as a Panel, it relies on an XML configuration file to store file paths, readable text copy, and editable text configuration (*see the Panel Developer's Guide*). For the example game, this is located in **/xml/panels/miniGameExample.xml**. Create a new file for your game at **/xml/panels/MyGame.xml**. Modify the file paths, text copy, and configuration to suit your game.

6.2.4 The Minigame Script Package

The root class for the example minigame exists in:
/src/com/disney/fairies/example/ExampleMiniGame.as.

Open this file to get a sense of how a talent minigame script looks. This class extends MiniGameBase and overrides several *event methods*:

- `onInitGame()` sets up the game content
- `onStartGame()` handles when game-play actually begins
- `onStopGame()` handles when game-play stops
- `onDestroyGame()` releases all resources when the game is over

Additionally, this class calls several framework *service methods* to perform intermediate subroutines such as reporting scores, manipulating the heads-up display, getting a convenient scrollRect for the content area, etc.

> *Refer to the API Reference section for a complete list of all event methods and service methods.*

Create a new class – **/src/com/disney/fairies/mygame/MyGame.as** – along with any necessary helper classes in the same package. Edit the scripts with your preferred IDE (*e.g.* Flash CS3, Flex Builder, Flash Develop, TextMate, etc).

Once again, scripts must limit themselves to only basic Flash and the minigame API. To ensure proper integration with the MMO client, contact Schell Games and SilverTree Media to discuss and request additional API features.

6.2.5 The ANT Build entry

/src/build.xml contains a panel build target for the example minigame:

```
[…]
<target name="exampleMiniGame">
<antcall target="panel">
<param name="swf" value="miniGameExample.swf"/>
<param name="class" value="com.disney.fairies.example.ExampleMiniGame"/>
</antcall>
</target>
[…]
```

Copy this target and change the name to **myGame**, the output SWF filename to **myGame.swf**, and the output SWF root class to **com.disney.fairies.mygame.MyGame**. Invoke this target at a shell prompt with the following commands:

```
cd path/to/working/folder
ant myGame
```

Alternately, you can write a shell script to do this. For example, in Windows create a file called "build.bat" and type "ant myGame" in this. Now invoke the build by simply double-clicking on this shell script icon in Windows Explorer.

6.2.6 Running the MiniGame

Open **xml/panels/miniGameContainer.xml**. This file contains the available minigames, for example:

```
<panel id="exampleGame" gameId="0"
transition="none">miniGameExample.xml</panel>
```

Add a new entry for your game and relaunch index.html.

```
<panel id="myGame" gameId="1" transition="none">myGame.xml</panel>
```

Your game should now appear in the list of available games.

6.3 Content Requirements

The content FLA must contain the five primary game screens: `title`, `instructions`, `prize`, `game`, and `bg`. The example minigame displays these symbols on the stage with the appropriate instance names. Some of these symbols also contain specific content. The remainder of this section describes the MovieClip symbols and their required content.

`title`

The title screen is the first symbol that appears when the game executes. The title screen contains a keyframed animation that displays it on the stage. The first frame of the animation is the symbol's default inactive state, and the last frame of the animation is the symbol's displayed state.

The title screen contains the title of the game and a MovieClip symbol instance, `btnContainer`. Inside `btnContainer` there are three MoveClip instances that act as rollover buttons: `playBtn`, `helpBtn`, and `quitBtn`. The player can click on `play` to start the game, `help` for instructions, and `quit` to exit the minigame. There are three states to each of these buttons: `up`, `down`, and `over`, represented as three labeled frames inside the button instances.

It is very important that `btnContainer` is keyed on every frame of the animation, so that the script does not lose its reference.

`bg`

The background symbol is used to create a common background for all minigames. Any appropriate background bitmap may be placed into this symbol. It will be displayed behind all the other screens, but can be toggled off during gameplay using a service method (*see API Reference*).

`instructions`

The instruction screen provides the rules for the minigame after the help button is clicked on. The instruction screen also contains a MovieClip symbol instance called `btnContainer`. The btnContainer contains four rollover buttons: `playBtn`, `backBtn`, `continueBtn` and `quitBtn`. The play and back buttons are displayed if the instructions are pulled up outside of gameplay and take the user into gameplay, or back to the title screen. The continue and quit buttons are displayed while the game is paused, and will unpause the game, or exit to the rewards screen.

`prize`

The prize screen is displayed after the minigame is over. The prize MovieClip contains a celebratory message and a `btnContainer` MovieClip. Within button container, there is a `containerBox` MovieClip that displays the prize, and two buttons: `replayBtn` and `quitBtn`. The replay button restarts the game and the quit button exits the game. The containerBox clip needs to contain a rectangle with specific dimensions (*TBD*).

`game`

The game MovieClip holds the main content of the game. All bitmaps and animations that are not included in the other display screens are placed in the game symbol. By naming the MovieClip instances within game symbol, developers can manipulate its content in ActionScript. At runtime, this MovieClip will be accessible using the `gameClip` getter.

7 API Reference

Overridable methods for handling specific game events:

`onInitGame():void`
called while the game is loading

`onStartGame():void`
handles when game-play actually begins

`onStopGame():void`
handles when game-play stops

`onPauseGame():void`
handles when the game is paused, in case event listeners need to be removed

`onUnpauseGame():void`
called when the game is unpaused, so that event listeners can be reregistered

`onDestroyGame():void`
called when the user is exiting the game, to release all resources

Service Methods that are provided to perform common game tasks:

`unrealize():void`
This method can be called during onInitGame() in order to halt the loading sequence from completing until some additional asynchronous request has completed (e.g. loading external files, prerendering bitmaps, etc).

`realize():void`
This method is called in an asynchronous event handler to indicate that the loading sequence should continue as notmal. It should_not_ be called inside of onInitGame().

`get config():XML`
Returns the <config> XML element from the main game configuration.

`getDefinition(linkageName:String):Class`
Retrieve a Class object for the given linkage name in the context of the content ApplicationDomain.

`getFromLibrary(linkageName:String):MovieClip`
Retrieve a MovieClip from the content's Library for the given linkage name. For instance, if in flash you would write:

```
var fooSymbol:MovieClip = new Foo();
```

You would now write:

```
var fooSymbol:MovieClip = getFromLibrary("Foo");
```

`getSoundFromLibrary(linkageName:String):Sound`
Retrieve a sound object from the content library.

`getBitmapDataFromLibrary(linkageName:String):BitmapData`
Retrieve a BitmapData object from the content library.

`getCopy(tag:String):String`
Retrieve a readable text string from the <copy> element in the main game configuration.

`get gameClip():MovieClip`
All gameplay content should be added under this MovieClip. Do not add content directly to MiniGameBase.

`get params():MiniGameParams`
Retrieve information provided to the game by the framework. The game should not poll the framework directly, otherwise MiniGames will cannot be reliably integrated with the MMO game.

`reportScore(value:int):void`
Queue a score to report to the server when the game terminates. Will be used to compute prizes server-size.

`endGame(event:Event = null):void`
Request to end gameplay and move to the rewards screen.

`enableViewportScrollRect(flag:Boolean = true):void`
MiniGames are displayed inside a diecut, and cannot spill out beyond the ordinary game bounds. This methods adds a viewport scrollRect to gameClip to ensure that it stays clipped.

`enableBackground(flag:Boolean = true):void`
Used to toggle the common background during gameplay.

HUD Methods provided to control the Heads-Up Display reliably.

Heads-Up Display Elements appear above the diecut, are built when gameplay begins and are destroyed when gameplay ends. All HUD elements must contain a transition animation on their main symbol timeline, and are accessed using subclasses of the HUDElement class. The MiniGame API make available several standard HUD elements, and also provides a mechanism for adding and removing generic subclasses of HUDElement as well.

```
hudAddTitle(head:String = "", subhead:String = ""):HudTitle
hudAddGenericElement(elem:HudElement):void
hudAddInterLevelDialog(message:String = ""):HudInterlevelDialog
hudAddGameOverDialog(message:String = ""):HudGameoverDialog
hudAddLeftImageAccent():HudAccentedHeading
hudAddRightImageAccent():HudAccentedHeading
hudAddLeftScoreCard():HudHeading
hudAddRightScoreCard():HudHeading
hudRemoveTitle():void
hudRemoveGenericElement(elem:HudElement):void
hudRemoveInterlevelDialog():void
hudRemoveGameoverDialog():void
hudRemoveLeftImageAccent():void
hudRemoveRightImageAccent():void
```

```
hudRemoveLeftScoreCard():void
hudRemoveRightScoreCard():void
```