RPG Playground Manual Documentation

Release latest

November 02, 2016

Contents

Here is the complete documentation on how to create your own Role Playing Game using RPG Playground.

If you just want a quick 3 minute introduction, take a look at the YouTube video at https://www.youtube.com/watch?v=xjBoXxkXauE.

CHAPTER 1

Introduction

1.1 What is RPG Playground?

With RPG Playground you can make your own multi-platform Action RPG. It runs in your internet browser and you can try it at http://rpgplayground.com.

A great story and a lot of imagination is all you need. Programming, graphics and music is provided by RPG Playground.

Remark that RPG Playground is still in development, so a lot of features are missing. However, you can already create your own adventure. For an overview of what is already available, and what will be added in the near future, take a look at the *next section*.

RPG Playground is created by me, Koen Witters. If you have questions or want some new feature, just email me at koen@koonsolo.com.

1.2 Features

Here are the features of the RPG Playground game editor:

- Easy to use map editor to build your world and indoor scenes
- Add NPC's and monsters to your world
- Create complex conversations and cut-scenes, using a very simple scripting language

And here is a quick overview of the features of your game:

- Control the main hero
- Outdoor and indoor levels
- Talk to NPC's
- Play as a Flash game in any web browser

Since RPG Playground under development, following features will be added in the near future:

- Action RPG combat system
- Item management
- Import your own graphics for tilemaps and characters

1.3 What is so different about RPG Playground?

The normal workflow of game maker tools is to start up the editor, adapt tilemaps, behaviours, etc, and then generate the game. Once the game is generated, you can test it. If you want to change something, you exit the game and go back to the editor. This kind of workflow can be cumbersome and time consuming.

RPG Playground reverses this workflow. You immediately start inside your (emtpy) game. When you want to make some changes, just open the project panel on the left, and make your changes straight into the running game. There is no real separation between editor and game anymore, you design your game while playing it.

There are several advantages using this approach:

- You see the impact of your changes immediately in the game.
- It's more fun to work inside your game than inside a boring editor.
- No need to search for the resource to adapt, you can just click on it and start editing.
- Continue playing where you left off after your changes.
- No need to compile/build your game before you can play it.

1.4 Supported platforms

RPG Playground itself runs on Adobe Flash, and your game will also run on it.

In the future, you should also be able to export your game as a stand alone version for the following platforms:

- Windows
- Mac
- Linux
- iOS
- Android
- Adobe Flash

CHAPTER 2

Interface Overview

2.1 Defining a game

The best way to look at a video game is think of it as a small world. At the base it contains the world itself where all objects live. This world has some rules that all objects must obey.

The objects themselves are divided into static objects that don't interact, and objects that can interact.

A sandbox game would fit the above description perfectly, but most games also contain more than the game world alone. They guide the player through a fascinating story (or multiple stories). So in that sense, a game could also be thought of as a movie, were the player experiences an adventure going from one situation to the next.

Now what does this all have to do with RPG Playground? Well, when designing your game, it's important to understand the fundamental concepts of how your game is composed. If you understand the general philosophy behind an RPG Playground game, it's easier to create your game exactly as you want it.

2.2 General Concepts

RPG Playground borrows a lot of terms from film and theater to describe the different concepts that make up a game.

First of all, the game world described above is called a *map*. The interactive objects that live in that world are called *actors*, and the static objects are called *scenery*.

The collection of the map, scenery and actors is called a *scene*.

2.2.1 Scene

A scene is best described by the old-school game term "level". It contains everything inside a single part of the game, which includes:

- A map, such as a tilemap representing grass, dirt, water, ...
- Scenery such as houses, trees, ...
- Several actors, such as the main hero, enemies, ...

A game is composed out of different scenes. Most of the time, only 1 scene is displayed at a time. But it is possible that multiple scenes are shown at the same time. This can be the case when using a game head-up-display for example. The head-up-display or HUD is a different scene, but is shown together with the main level scene.

2.2.2 Map

A map in our context, is the place or world where the actors live. RPG Playground uses a tilemap as map, but you could imagine other maps such as a side-scroller maps or isometric tilemaps.

The map partly defines how actors can move around, what their position is, etc.

2.2.3 Scenery

A map can contain static objects, and these are called scenery. Things such as trees, bridges, houses, etc. are all scenery. They contain no interaction, although they can be animated.

The difference between scenery and actors is that scenery doesn't interact with the player or with the game world.

2.2.4 Actors

Actors are the interactive part of the game. Examples are the main hero, enemies, animals, etc. . But remark that interactive objects such as doors, treasure chests, moving cars, etc. are also described as actors. They play an interactive role inside our game, and are therefore called actors.

2.3 The Project Pane

Now that you know the basic components of an RPG Playground game, let's see how we can work with it in practice.

When your game is running, on the top-left there is an *open project pane* button. When clicking it, the project pane will open on the left side of the screen. You can close this dialog and resume your game by pressing the *close project pane* button on the top-right of the project pane.

The project pane contains the following tabs:

- Map
- Scenery
- Actors
- World tree
- Project

The project panel also contains a *maximize button* on the top-right, to maximize the game onto your entire computer screen.

2.4 The Map Tab

The map tab allows you to manipulate the map. Currently you can resize your game level with it.

2.5 The Scenery Tab

The scenery tab allows you to add and remove scenery such as trees, buildings, tables, etc. to your level.

2.6 The Actors Tab

With the actors tab you can add and remove game characters and monsters. You can also change their properties, and specify conversations and other actions that can be done.

More details in section *Actors Tab*

2.7 The Project Tab

The project tab allows you to save your game, and afterwards load it back. There is also a help button that can show a tutorial video, or point you to this manual.

CHAPTER 3

Actors Tab

3.1 Tokens Component

You can clear all tokens received by the hero by pressing the *restart* button on the right of the *Tokens* component title.

Tokens are set using screenplays, see *hero receives token "token"* for more details.

CHAPTER 4

Screenplays

When designing your game, you will need to add specific game functionality. For example when you open a treasure chest, step on a trap, or when you bump into a character. You want to be able to have conversations, receives items, etc. When a new scene starts, you might want to show a cutscene that reveals a part of the story. In any case, all of these things can be created with screenplays.

Screenplays are currently specified in plain text. But don't worry, it is much easier than any programming or scripting language. Just take a look at the example screenplay below:

```
hero says "Hi mom!"
mom says "Hello sweetheart. Are you going into the forest?"
hero says "Yes"
mom says "Be carefull, alright?"
if hero lacks token "dagger"
    mom says "Take this dagger just in case."
    hero receives token "dagger"
dad says "And be home before supper!"
```

It reads just like a film script doesn't it? Well, thats the whole purpose, to keep it simple and readable so anyone can work with it.

Remark that only a small subset of the complete screenplay features are currently implemented. More actions will be added soon. To get an idea on what will be supported in the future, take a look at the following script:

```
screen shows text "You enter the library" with icon book
professor says "Welcome my friend, if you have questions, you know where to find me"
continue
professor says "Hi again. The end of the world is near!"
when player chooses "How do you know this is the end of the world?"
    professor walks to tile 3, 4
    professor says "Take a look in this book, the prophecy is clear"
    if hero has token "prophecy book"
        hero says "I already have that book"
    else
        hero says "Let me see!"
when player chooses "This is not the end, it's just an earthquake."
    professor says "Unbeliever!"
    professor walks to tile 0, -6
```

4.1 Actions

A screenplay consists of a sequence of actions. An action is always of the form <*actor*> *action*, where <*actor*> is an actor that supports the given action.

Here is the list of actions that can be used inside a screenplay:

- character says "your text" (*more details*)
- hero receives token "<token>" (*more details*)
- hero loses token "<token>" (*more details*)
- if hero has token "<token>" (*more details*)
- if hero lacks token "<token>" (*more details*)

4.1.1 character says "*your text*"

This action opens a conversation dialog displaying *your text*. See the example below

Example screenplay:

```
hero says "Do you know the way to Lotheritus?"
merchant says "No, but maybe you can ask around in the Inn."
```

The formal specification of this action is:

```
<actor> says "<your text>"
```

<actor> *hero* or any of the NPC's

<your text> The text you want to display in the conversation dialog. Make sure this is not too long so that it doesn't fall outside of the dialog.

4.1.2 hero receives token "*token*"

This action gives the hero a token, which can be checked in screenplays with *if hero has token "token"* or *if hero lacks token "token"*. Tokens work across different scenes, so you can set a token in one scene and check for it in another scene.

When playtesting, you can clear all tokens by selecting the hero entity and clicking the *restart button* next to the *Tokens* title. See *Tokens Component* for more info.

Example screenplay:

```
if hero lacks token "knows lisa"
    lisa says "Nice to meet you, I'm Lisa."
    hero says "Nice to meet you too, my name is Bart."
    hero receives token "knows lisa"
else
    hero says "Hi Lisa, nice to see you again"
```

The formal specification of this action is:

```
hero receives token "<your token>"
```

<your token> The name of the token that the hero should receive. Might have already been received.

4.1.3 hero loses token "*token*"

This action removes a token received by the hero with *hero receives token "token"*.

Example screenplay:

```
if hero has token "book of doom"
    hero says "Here is the book, I don't want it anymore."
    hero loses token "book of doom"
    lisa says "Thanks, I guess..."
```

The formal specification of this action is:

```
hero loses token "<your token>"
```

<your token> The name of the token that the hero should lose. It might not be set yet.

4.1.4 if hero has token "*token*"

Checks if the hero received a certain token, and if so, does the actions in the sub-block right after this action. An *else* action can be used after this action, which will run in case the hero didn't have that token.

Remark that only the *hero* actor supports tags!

Example screenplay:

```
if hero has token "knows lisa"
    hero says "Hi Lisa, nice to see you again"
else
    lisa says "Nice to meet you, I'm Lisa."
```

```
    hero says "Nice to meet you too, my name is Bart."
    hero receives token "knows lisa"
```

The sub-block of the *if* is a list of actions that are run when the hero received the token. All actions in this sub-block must contain 4 extra spaces in front of each action.

The formal specification of this action is:

```
if hero has token "<your token>"
    <actions when true>
else
    <actions when false>
```

<your token> The name of the token that is checked.

<actions when true> A sequence of actions that are run when the hero has token *your token*. All these actions need to be proceeded with an extra 4 spaces relative to the *if*.

else This *else* is optional and so can be skipped.

<actions when false> A sequence of actions that are run when the hero doesn't have token *your token*. All these actions need to be proceeded with an extra 4 spaces relative to the *else*.

4.1.5 if hero lacks token "*token*"

Checks if the hero didn't receive or lost a certain token, and if so, does the actions in the sub-block right after this action. An *else* action can be used after this action, which will run in case the hero does have that token.

Example screenplay:

```
if hero lacks token "knows lisa"
    lisa says "Nice to meet you, I'm Lisa."
    hero says "Nice to meet you too, my name is Bart."
    hero receives token "knows lisa"
else
    hero says "Hi Lisa, nice to see you again"
```

The formal specification of this action is:

```
if hero lacks token "<your token>"
    <actions when true>
else
    <actions when false>
```

<your token> The name of the token that is checked.

<actions when true> A sequence of actions that are run when the hero doesn't have token *your token*. All these actions need to be proceeded with an extra 4 spaces relative to the *if*.

else This *else* is optional and so can be skipped.

<actions when false> A sequence of actions that are run when the hero has token *your token*. All these actions need to be proceeded with an extra 4 spaces relative to the *else*.

4.2 Errors

When typing screenplays, writing errors is pretty common. Therefore, any errors will be reported at the bottom of the screenplay edit dialog.

Below is a list of all errors, accompanied with a more detailed explanation.

4.2.1 Entity '...' not found

Your entity name doesn't exist in this scene. Check to see if the name is correct.

4.2.2 Entity '...' has no action '...'

Your specified entity doesn't have given action. See *Actions* for a list of all available actions.

4.2.3 An indent of X spaces is not allowed, use a multiple of 4 spaces

A single indent in the code is represented by adding 4 spaces in front of the action. Such indents are used for example with *if* or *else* actions. This error says that you don't have a multiple of 4 spaces in front of your action.

4.2.4 Action '...' needs to follow an 'if' action

You are probably using an *else* action without first specifying an *if* action. See the example in *if hero has token "token"* on how to use it properly.

4.2.5 Previous action doesn't support an indent block

Using 4 spaces as indent is only allowed right after an *if* or *else* action.

4.2.6 Cannot indent more than 1 level

A single indent is 4 spaces, you are using too much spaces for this indent.

CHAPTER 5

Frequently Asked Questions

Got any questions? Ask them in the RPG Playground forums

CHAPTER 6

Appendix A: User profiles

RPG Playground needs to be both super user friendly, and powerful. This is not an easy combination, because the more things you can to with a product, the harder it is for you to understand and start using it right away.

But there is a way that RPG Playground is able to combine the two, and this is by using multiple abstraction layers. If you are a new user, you start working in the top layer, and don't need to know anything about those lower layers. And when you gain more experience, you can work your way through the lower layers, and are able to customize everything inside your game.

The following layers are defined:

Game designer You design levels, add characters to it, and add cutscenes, dialogues, etc. You work with scenes, stages, scenery and actors. Define new functionality with screenplays

Gameplay scripter You can define new actors and add or remove behaviors. You can even script new behaviors for actors. Define new functionality with LUA scripts.

Artist You draw your own tiles, characters and objects, even with animations. and can import them as a resource into your game. Or you compose your own music and use that for the game Edit with external tools, import into RPG Playground library.

Coder You dive into the game engine source code and can program your own functionality just the way you want it. Haxe source code editing.

As already mentioned, RPG Playground is far from finished. So for the moment, the focus is purely on the top level. The current functionality tries to support everything that a game designer needs.