Flat Hunt Developer Guide

Ursina Caluori
ucaluori@student.ethz.ch

August 21, 2005

Contents

TRAFFIC [2], *Touch* [1] and *Flat Hunt* is software that hopefully makes learning to program more fun and more interesting for you. *TRAFFIC* is a library that supports the reading and display of public transportation systems. A library is a piece of software whose functionality can be used by other software. *Flat Hunt* is an application that uses the *TRAFFIC* library to model a city map. For the visualization the *EiffelMedia* Library (formerly known as *ESDL* [6][5]) is used. It is a strategy game, similar to Scotland Yard, but with a different background story (for more information about the story and gameplay of *Flat Hunt*, read the Flat Hunt User Guide). *Touch* is also an application that uses the *TRAFFIC* library. For more information on *TRAFFIC* and *Touch*: see [2] and [1] or visit this website.

This document describes how *Flat Hunt* is built, what classes are important and highlights some of their features.

1 Getting Started

What you need for running Flat Hunt...

1.1 Requirements

1.2 Installation

2 Design

2.1 Overview

When opening *Flat Hunt* in EiffelStudio, the cluster view in the bottom left corner of EiffelStudio shows many clusters. For you only the top-level clusters *Traffic* and *Flat_hunt* are important.

To remove complexity, *Flat Hunt* is structured in three clusters (see Figure 1): *Model*, *View* and *Controller*. In each cluster, there are several classes, and sometimes there are subclusters.

2.2 Controller cluster

Cluster Controller is the fundamental cluster in *Flat Hunt*. Here are the classes that "control" the actions. They make sure that the displayer classes in cluster View display the proper information, which they get from the Model classes. For

example, feature prepare in class GAME controls the display update by calling current_player.displayer.display_after_move.

2.3 Model cluster

In the cluster Model, there are two important parent classes: Class PLAYER and class BRAIN. PLAYER is the parent of FLAT_HUNTER and ESTATE_AGENT, and BRAIN is the parent of HUMAN, FLAT_HUNTER_BOT and ESTATE_AGENT_BOT. These Model classes describe the internal representation of "real world" objects. Here is a description of some of these classes.

2.4 View cluster

This clusters job is to make sure that the user sees what is going on. It includes all scenes and menus, as well as displayers for the game players and status information.

2.5 Other

3 The States of the Game

3.1 Overview

Every game has at least two states: playing and game over. *Flat Hunt* has six states in total; three playing states and three game over states (see Figure x). These game states are defined in class GAME_CONSTANTS:

```
   Agent_stuck, Agent_stuck, Agent_caught, Agent_escapes,
Prepare_state, Play_state, Move_state:  INTEGER is unique
```

3.2 Game Loop

For each player in each round in *Flat Hunt*, the game goes through the following states: Prepare, Play and Move. In addition, there are three game over states: Agent_stuck, Agent_caught and Agent_escaped.

Prepare If the game is in this state, the current player gets a red circle and the possible moves are calculated and displayed. If the current player is the estate agent, and there are no possible moves, the agent is stuck and thus the game is over (state Agent_stuck). If that is not the case, the game goes in state Play.

Play In this state, if the current player is played by a human, the game waits until the human player clicks on one of the places that are highlighted. If the player is controlled by an artificial intelligence, then the best of the possible moves is calculated. The game then goes in state `Move`.

Move In this state, the move selected in state `Play` is performed. After the move, the game checks if the player hits the place of the estate agent. If that is the case, the game goes into state `Agent_caught`. If the agent did not get caught, and the round number is greater than 23, then the estate agent is the winner and the game goes into state `Agent_escaped`. If none of the above is the case, then it's the next player's turn and the game loop starts again in state `Prepare`.

4 Guided "Walk-Through"

What happens when you start Flat Hunt*? In this last chapter we will go step-by-step through a typical* Flat Hunt *game. However, because there are lots of details involved, we concentrate on the more important steps...*

1. start game blabla
2. do something
3. end game blabla

5 Legal Stuff and Thanks

This document is based upon its prior version, which was written by Michela Pedroni and Marcel Kessler (thanks!). All graphics for the game were designed by me and Photoshop. The code of *Flat Hunt* is based on its prior version [**?**], which is mainly the work of Marcel Kessler. Major parts had to be rewritten by me though.

Thanks to Michela Pedroni for her assistance, all my predecessors for their work, Till G. Bay (and others) for the *EiffelMedia* Library (formerly *ESDL* [6][5]) and Bertrand Meyer for the *Eiffel* language.

References

[1] Roger Küng. *Touch User Guide*. ETH Zurich, 2005.
http://se.inf.ethz.ch/projects/roger_kueng

[2] Sibylle Aregger. *Redesign of the TRAFFIC library*. ETH Zurich, 2005.
http://se.inf.ethz.ch/projects/sibylle_aregger

[3] Rolf Bruderer. *Object-Oriented Framework for Teaching Introductory Programming*. ETH Zurich, 2005.
http://se.inf.ethz.ch/projects/rolf_bruderer

[4] Marcel Kessler. *Exercise Design for Introductory Programming. "Learn-by-doing" basic OO-concepts using Inverted Curriculum*. ETH Zurich, 2004.
http://se.inf.ethz.ch/projects/marcel_kessler

[5] Benno Baumgartner. *ESDL - Eiffel Simple Direct Media Library*. ETH Zurich, 2004.
http://se.inf.ethz.ch/projects/benno_baumgartner

[6] Till G. Bay. *Eiffel SDL Multimedia Library (ESDL)*. ETH Zurich, 2003.
http://se.inf.ethz.ch/projects/till_bay