Testing Document

Planetary Pitstops

Nathan Moore

October 7, 2013

Revised:

October 9, 2013

Section 1 – Introduction

This will be a retro style side scrolling action platformer reminiscent of games from the Nintendo Entertainment System and Super Nintendo era. The player needs to complete a series of levels by collecting all of the energy crystals needed to power the astronaut's space ship – allowing him to fly to the next level and get a little bit closer to home. This game is primarily for entertainment purposes and would be most enjoyed by people who enjoyed playing games like Megaman and Metroid. See the design document for a more complete description of the game.

This testing document will go over some of the methods that will be used to test the game to make sure it will function properly. This will involve doing things like logging, stepping through the program with a debugger, exception handling, a debug mode while the game is in play, and possibly unit testing.

Section 2 – Overview of the Game

A lot of complexity may arise from the architecture of the game engine, since it will be designed with flexibility in mind. There is no knowing exactly how all the interacting components will behave together, but effort will be put into making all of it work well together.

The game will end when the player either completes all the levels and reaches the “wins” the game or by choosing to quit playing. The game is automatically saved at the beginning of each level and the player can save his/her progress during play by stepping on one of the Emergency Medical Teleporters. This save file can be reloaded at the start of the next play session or it will be reloaded if the player's health is depleted by touching too many enemies or hazards.

This will be a graphical game with images, animation, sounds, and music. One potential source of error would be attempting to load an asset that doesn't exist. Refer to the design document for more information.

Section 3 – Top Level Failure Modes

Because the game engine will be flexible and largely data driven, there may be many sources of possible failure. The levels will be designed using the Tiled tile map editor, which allows for marking where objects and regions should be along with data values those might use. One possible source of

error would be by specifying an invalid value or by failing to accurately define a needed parameter (for example, by making a typo or, simply, by omitting it from the parameter list – though default values should compensate for this.)

The level of control the player has over the inputs fed into the game will be limited – directions, jump, and fire buttons. Problems may arise, however, by game objects or their components interacting in unexpected ways.

During development it would be better for the program to fail hard – breaking the program at a point of error and dumping relevant information. At release, though, it would be preferable for problems to be gracefully handled. For example, during development if I try to load an image that does not exist it would be useful to immediately be alerted of it, while a person is playing the game, though, it would be better to load in a missing image placeholder (like a white box with an X in it). This might look ugly to the player, but at least the game will be playable.

Section 4 – Modules and Testing

Pretty much no testing will be done to make sure external modules are functioning properly, but the game will have a lot of its own modules that should be fully tested. The game will mostly depend on PyGame, but modules for loading Tiled tile maps will be considered. If those do not seem to meet the project's needs, a custom Tiled map loader may need to be written.

Some modules that come with Python that might be beneficial for testing are “logging,” for logging error messages and warnings, and “unittest,” for designing test cases to make sure everything passes inspection.

PyCharm's debugger seems to do a pretty good job, so that will be used for tracking down errors. Of course, the good old fashioned print style debugging will also be used from time to time.

Section 5 – Top Level Testing

Logging will be done using the logging module whenever anything unexpected happens, like trying to load missing assets, invalid parameters are entered, or an exception is caught. There may also be times when warnings are logged when only minor problems occur.

The unittest module will be used to design unit tests for most of my game's modules. These will be

put in the “tests” folder in the main source folder. Unit testing will be done to check if the results and behaviors of the functions and class methods are correct.

PyCharm's debugger will be used when unexpected behaviors occur, like crashes or unusual results. Breakpoints will be placed in around where the problem is suspected to be occurring and the program will be stepped through to determine what is going wrong.

Gameplay testing will be done not only by me, but by whoever I can get to play the game. This might include class mates, family members, friends, and random strangers on the internet. Their feedback will be used to try to make improvements to the game in an attempt to have it be more fun and enjoyable. They should also report whatever errors and bugs they encounter.

Section 6 – Implementation Plan and Timeline

Testing will be done throughout development, but extra time near the deadline should be spent on tweaking, adjusting, and debugging whatever problems may be found. The goal should be to release a stable application free of all the major and obvious problems. Some smaller problems may slip by undetected, but hopefully nothing game breaking will get through.

Because of this interleaved programming and testing a definite schedule would be difficult to clearly define. The core engine should be completed and fully tested relatively early so there will be time available for game play testing.

Section 7 – Bibliography

[1] David Sale (2013, Feb.) “Beginning Test-Driven Development in Python” [Online]. Available: http://net.tutsplus.com/tutorials/python-tutorials/test-driven-development-in-python/

[2] Python Documentation “25.3. unittest – Unit testing framework” [Online]. Available: http://docs.python.org/2/library/unittest.html

[3] “TMX Map Format” [Online]. Available: https://github.com/bjorn/tiled/wiki/TMX-Map-Format

Section 8 – Work Assistance Statement

This page was written only with the assistance of the template file we were given.