Proof Of Concept

Cole Blanchard, Ratna Emani, Akshay Mantha

2 October, 2015

Revision History

Developer	Date	Change	Revision Number
Ratna Emani	December 5 2015	EDIT: Will a part of the implementation be difficult? EDIT: How will we overcome these risks?	4
Cole Blanchard	October 2 2015	How will we overcome these risks?	3
Akshay Mantha	October 2 2015	Will a part of the implementation be difficult?	2
Ratna Emani	October 2 2015	Will portability be a concern?, Formatting	1

1 Will a part of the implementation be difficult?

Zop was originally implemented in CoffeeScript; a programming language that trans-compiles to JavaScript. Our implementation of Zop will be in Python programming language - thus, understanding the original code and translating it into Python will be a sizeable task. One reason we are taking this risk is because we are more comfortable with Python than we are with CoffeeScript/JavaScript. Time is a constraint for this project as we have only eight more weeks to reproduce the entire project and we cannot learn an entire language in such a short period of time. Another reason to choose to implement Zop in Python is the original code itself. The native developer has written the source code in a very inefficient manner, making the code hard to understand for anyone else. The code is not commented and there is no proper modularization. More importantly we would like to take this opportunity to create a detail-oriented learning experience for new developers and programmers interested in the use of Python and Pygame. Hence, we believe taking the risk of implementing Zop in Python will be a fruitful experience, albeit a bit difficult one.

2 Will portability be a concern?

The game's original creator chose to use Coffee Script, Java Script and HTML to program Zop. It was posted on GitHub for other developers and gamers to access and play. This makes the game very portable, and easy to access. When we were assigned the task to redevelop the game, we chose to program Zop in Python using Pygame. Pygame is a python wrapper for SDL, written by Pete Shinners. Pygame allows developers to write games or other graphical applications in Python on all SDL supported platforms (Windows, Linux, Mac and others). It is this transition in the choice of the programming language that scarified the games portability. In order to start the game from source, the user must run the main.py using Python 2.7 (32-bit), and Pygame 1.9.1.

3 How will we overcome these risks?

In order to overcome the risk of the implementation we will demonstrate our ability to organize the code well so that if future developers were to work on the project, they would have the ability to easily read and understand the code that is in front of them. Simply commenting the code and using meaningful variable names with the additional help of thorough documentation should be a sufficient solution to this risk. Concerning the modularization of the code, we can demonstrate creating separate classes for certain parts of the game (game board, game pieces...) to make the code easier to navigate. ~~Portability-wise, we will need to negate this risk by providing the user with a fully functioning, bug-free game as well as potentially implementing newer and fresher features to the game such as power-ups and levels in order to attract players to actually download and play the game.~~ When it comes to the trade-off between the portability versus the modularization of the game. We as a team

came to the conclusion that it will benefit both the group and the developers to have a fully functioning, bug-free, detail-oriented program rather than a easy-to-access game.