SE 3XA3: Test Plan
PyCards

Team 2
Aravi Premachandran premaa
Michael Lee leemr2
Nikhil Patel patelna2

October 31, 2016

Contents

List of Tables

List of Figures

Table 1: **Revision History**

Date	**Version**	**Notes**
October 27	0.1	Rough Draft

1 General Information

1.1 Purpose

The purpose of conducting testing is twofold. While testing cannot prove correctness or the absence of bugs, it can be useful for finding instances of incorrect behaviour. By ensuring that testing is done in a traceable and repeatable manner (namely through automation), defects that are uncovered can be traced, isolated, and addressed. With automation, testing can be performed throughout the life cycle of the product with very little overhead.

The other reason for conducting testing is to demonstrate to the client that our product is reliable, robust and meets the requirements that were set forth (using fit criterion or some other measure of degree of fulfilment).

1.2 Scope

PyCards is a collection of card games implemented as a desktop application. As with any software program, it is important that it undergoes various iterations of testing throughout its product lifecyle. Our development team is using a number of different test types, including functional, structural, and unit tests, static and dynamic, manual as well as automated. While automated testing is largely preferred for reasons such as greater traceability, reproducibility, and efficiency, testing will also need to be done manually, especially for validating non-functional requirements. Thus, the scope of testing for this product includes functional, structural, and unit tests, static and dynamic testing, and manual and automated testing.

1.3 Acronyms, Abbreviations, and Symbols

Table 2: **Table of Abbreviations**

Abbreviation	**Definition**
Abbreviation1	Definition1
Abbreviation2	Definition2

Table 3: **Table of Definitions**

Term	**Definition**
Widget	A functional component of a graphical user interface. Usually consists of a visual component and a callback
Callback	A method to be executed upon interaction with a widget
Bijection	A one-to-one mapping of elements in one set to another

1.4 Overview of Document

This document provides a detailed description of the testing our development team has deemed necessary for the software product. The tests are categorized and subdivided based on the type of testing, the scope of said categories, and the purpose and application of the tests (ie. validating the fulfillment functional or non-functional requirements).

This document is subject to revision throughout the expected life of the product. It is not expected that many deletions or shrinking of the test sets will occur; however, additional testing will likely be prescribed and document as the product is developed and matures.

2 Plan

This section details the testing process prescribed for the software product, including but not limited to the testing team, schedule, techniques, and technologies.

2.1 Software Description

2.2 Test Team

- Aravi Premachandran
- Michael Lee
- Nikhil Patel

2.3 Automated Testing Approach

The testing team will be applying automated testing for a subset of the structural and static tests. In particular, unit tests will primarily be automated to increase reproducibility and efficiency, among other factors. It should be noted that in automated testing, only the execution and evaluation (of pre-defined criteria) is automated - in the event of failures or unexpected behaviour a member of the testing team will still be required to analyze the requirements, the code, and the test itself to determine where the inconsistency, if any, is located.

2.4 Testing Tools

- IDE - PyCharm
  - static, structural: syntax checking, reachability, adherence to coding conventions
- pylint / pyCheckers
  - static, stuctural: syntax checking, reachability, adherence to coding conventions
- unittest.py - built-in module for testing
  - dynamic, unit test
    - mock module - can be used for stubs and drivers, to isolate code

2.5 Testing Schedule

See Gantt Chart at the following url ...

3 System Test Description

3.1 Tests for Functional Requirements

3.1.1 Event Handling

Key Bindings

1. KB1
   Type: Functional, Dynamic, Manual

   Initial State: Application instance that is capturing user input

   Input: Keyboard press of one of COMMAND_KEYS by user

   Output: Application perform the associated action from BOUND_ACTIONS. Please refer to **Table 4 Keymapping Configurations** for mapping of keys to actions

   How test will be performed: The application will be launched and once loaded, each bound key in COMMAND_KEYS will be pressed and we will observe whether the correct action from BOUND_ACTIONS executes

3.1.2 Widget Callbacks

1. WC1
   Type: Functional, Dynamic, Manual

   Initial State: Menubar widget waiting for user interaction

   Input: User clicks on a label in the menu bar

   Output: If the user clicks or hovers on a cascading menu it will expand to show all contained submenu labels. If the user clicks on a menu label it will perform its associated callback function

   How test will be performed: With full knowledge of what behaviour should result from clicking any of the menu labels, the user will choose a random subset of labels to click, and compare the expected and actual behaviour of the application

2. WC2
   Type: Functional, Dynamic, Manual

   Initial State: Toolbar widget waiting for user interaction

   Input: User clicks on a toolbar button

   Output: The application will execute the appropriate function based upon the toolbar button clicked. There is a bijection between the actions in BOUND_ACTIONS and toolbar buttons

   How test will be performed: The application will be launched and once loaded, each bound key in COMMAND_KEYS will be pressed and we will observe whether the correct action from BOUND_ACTIONS executes

3. WC3
   Type: Functional, Dynamic, Manual

   Initial State: Card widgets waiting for user click

   Input: Primary button click on card widget

   Output: If the card selection is valid (see Functional Klondike Requirements in PyCards SRS for selection constraints) the card is highlighted while the mouse button remains pressed and is redrawn to follow the cursor. If the card selection is invalid no visible changes are made to the window.

   How test will be performed: With a game in progress two cards will be clicked one 'valid' and the other an 'invalid' selection and we will observe to make sure only the 'valid' selection results in a card being highlighted

3.2 Tests for Nonfunctional Requirements

3.2.1 Interoperability

Operating System Compatibility

1. OSX
   Type: Structural, Dynamic, Manual

   Initial State: Executable for program is on target machine running a standard install of OS X OSX_VERSION.

   Input/Condition: User locates executable and launches program

   Output/Result: Either a successful launch or a message from Gatekeeper alerting user that program was created by an unidentified developer

   How test will be performed: Executable is launched via double clicking to test whether or not Gatekeeper feature will block normal launch. User will observe whether basic functionality such as input/output and rendering of application window are successful

2. OS2
   Type: Structural, Dynamic, Manual

   Initial State: Executable for program is on target machine running a standard install of Ubuntu UBUNTU_VERSION.

   Input/Condition: User locates executable and launches program

   Output/Result: Either a successful launch or a failure caused by incompatible packaging, dependencies, or other causes

   How test will be performed: Executable is located and then launched via terminal on the target system. User will observe whether basic functionality such as input/output and rendering of application window are successful

3. OSWIN
   Type: Structural, Dynamic, Manual

   Initial State: Executable for program is on target machine running a standard install of Windows WIN_VERSION.

   Input/Condition: User locates executable and launches program via double-click

   Output/Result: Depending on the system, either a successful launch or application crash due to missing VC++2008 binaries (required even after building executable)

   How test will be performed: Executable is located and then launched via double-click on the target system. User will observe whether basic functionality such as input/output and rendering of application window are successful

Portability

1. PR1
   Type: Structural, Dynamic, Manual

   Initial State: Executable for application is located on a removable USB drive

   Input/Condition: User launches program executable from a removable USB drive

Output/Result: The program launches exactly as if it had been launched on the same system from the internal hard drive.

How test will be performed: User inserts removable drive into a machine running a supported OS then executes the executable either via double-clicking or using a shell (ie. bash, command-line). User will observe whether or not the program successfuly launches and operates

3.2.2 Usability

Navigation

1. UN1
   Type:

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

2. UN2
   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

3.2.3 Product Integrity

Resource Loading

1. RL1
   Type:

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

2. RL2
   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

3.2.4 Security

System Permissions

1. SP1
   Type:

   Initial State:

   Input/Condition:

   Output/Result:

   How test will be performed:

2. SP2
   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

4 Tests for Proof of Concept

4.1 Area of Testing1

Title for Test

1. PC1
   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

Input:

Output:

How test will be performed:

2. PC2
   Type: Functional, Dynamic, Manual, Static etc.

   Initial State:

   Input:

   Output:

   How test will be performed:

4.2 Area of Testing2

...

5 Comparison to Existing Implementation

6 Unit Testing Plan

6.1 Unit testing of internal functions

6.2 Unit testing of output files

References

7 Appendix

7.1 Symbolic Parameters

- COMMAND_KEYS = {'N', 'P', 'U', 'R', 'D', 'H', 'F1', 'F2', CTRL+N, CTRL+P, CTRL+'}
- BOUND_ACTIONS = {newgame, pause, undo, redo, deal, help, quit}
- NUM_REDEALS = Unlimited
- FOUNDATION_BASE = Ace
- STACK_BASE = King
- NUM_DECKS = 1
- NUM_WASTES = 1
- NUM_FOUNDATIONS = NUM_SUITS = 4
- NUM_NORMAL_STACKS = 7
- OSX_VERSION = EL_CAPITAN
- UBUNTU_VERSION = 16.04.1
- WIN_VERSION = 10

7.2 Reference Tables

Table 4: **Keymapping Configurations**

COMMAND_KEYS	**BOUND_ACTIONS**
'CTRL+N', 'N', 'F2'	new_game
'CTRL+P', 'P'	pause_game
'CTRL+Q', 'Q'	quit
'CTRL+H', 'H', 'F1'	show_help
'U'	undo_move
'R'	redo_move
'D'	deal_card

7.3 Usability Survey Questions?

This is a section that would be appropriate for some teams.