

SE 3XA3: Test Plan PyCards

Team 2

Aravi Premachandran premaa

Michael Lee leemr2

Nikhil Patel patelna2

December 8, 2016

Contents

1	General Information	1
1.1	Purpose	1
1.2	Scope	1
1.3	Acronyms, Abbreviations, and Symbols	1
1.4	Overview of Document	2
2	Plan	2
2.1	Software Description	2
2.2	Test Team	2
2.3	Automated Testing Approach	3
2.4	Testing Tools	3
2.5	Testing Schedule	3
3	System Test Description	4
3.1	Tests for Functional Requirements	4
3.1.1	Event Handling	4
3.1.2	Widget Callbacks	4
3.2	Tests for Nonfunctional Requirements	5
3.2.1	Interoperability	5
3.2.2	Usability	7
3.2.3	Product Integrity	9
3.2.4	Security	10
4	Tests for Proof of Concept	11
4.1	Game Logic	11
5	Comparison to Existing Implementation	12
6	Unit Testing Plan	12
6.1	Unit testing of internal functions	12
6.2	Code Coverage Metric	13
7	Appendix	14
7.1	Symbolic Parameters	14
7.2	Reference Tables	14
7.3	Usability Survey Questions	15

List of Tables

1	Revision History	ii
2	Table of Abbreviations	1
3	Table of Definitions	2
4	Keymapping Configurations	14

List of Figures

Table 1: **Revision History**

Date	Version	Notes
October 27	0.1	Rough Draft
October 28	1	Initial Version
December 7	2	Revision 1 - Added Gantt chart link, abbreviations, unit test explanations

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 lifecycle. 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
VC++ 2008 binaries	Required dlls included in the Microsoft Visual C++ 2008 redistributable package
IDE	Integrated Development Environment
USB drive	Flash drive; Portable storage device that connects to a computer via a Universal Serial Bus interface.

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

PyCards is a collection of card games implemented as a desktop application. The implementation is written in pure Python. The application is to be made available both as a standalone executable and in a source version.

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, structural: syntax checking, reachability, adherence to coding conventions
- coverage.py
 - dynamic, structural: code coverage of the program
- 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 in [project schedule](#)

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 successfully launches and operates

2. PR2

Type: Structural, Static, Manual

Initial State: The VC++ 2008 binaries required for execution of the program are not found on the host system

Input: The program fails to launch on a Windows-based machine that does not have the VC++ 2008 redistributable package installed

Output: All of the users are able to successfully follow the provided instructions to install the VC++ 2008 redistributable package

How test will be performed: A group of randomly selected users are given this issue and will be prompted by our application with instructions for resolving it. The criteria for success of this test is that all users are able to follow the instructions, install the required package, and after doing so successfully launch the application

Tolerance and Limitations: This test is limited to being performed on machines running a Windows installation with minimum WIN_VERSION. If the success criteria is not met because of unforeseen errors users will be advised to download and run the application from source

3.2.2 Usability

Navigation

1. UN1

Type: Structural, Dynamic, Manual

Initial State: The existing implementation is running and idle on the user's machine, and is waiting for user interaction

Input/Condition: A group of users are asked to perform each of the actions defined in BOUND_ACTIONS

Output/Result: The majority of users successfully perform the different actions, completing each within in a period of under MAX_FIND_TIME

How test will be performed: A group of randomly chosen people, of different ages and unspecified level of technical expertise will be asked to perform the actions defined in BOUND_ACTIONS. The amount of time it takes them to perform each task will be measured. The test will be considered successful if the majority of users are able to complete each of the individual actions in under MAX_FIND_TIME

2. UN2

Type: Structural, Dynamic, Manual

Initial State: Users have completed and taken note of the results of the previous test. Our application, PyCards, is now running and idle, waiting for user interaction

Input/Condition: The group of users are asked to perform each of the actions defined in BOUND_ACTIONS

Output/Result: The majority of users successfully perform the different actions, completing each within in a period of under MAX_FIND_TIME and in time less than or equal to what was required to perform the same action using the existing implementation

How test will be performed: A group of randomly chosen people, of different ages and unspecified level of technical expertise will be asked to perform the actions defined in BOUND_ACTIONS. The amount of time it takes them to perform each task will be measured. The test will be considered successful if the majority of users are able to complete each of the individual actions in under MAX_FIND_TIME and in time less than or equal to time taken using the existing implementation

Playability

1. UP1

Type: Structural, Dynamic, Manual

Initial State: Existing implementation is running, with a game in progress

Input/Condition: A group of users are asked to play a game and rate it based ease of use using a scale from 1-5 where 5 is very user friendly

Output/Result: The majority of users give the game an average rating above 3

How test will be performed: A group of randomly chosen people, of different ages and unspecified level of technical expertise will be asked the questions defined in the Appendix and to give answers from a scale from 1-5. The scale maps as: 1 - Very hard to use, 2 - Hard to use, 3 - Some effort to use, 4 - Easy to use, 5 - Very easy to use

2. UP2

Type: Structural, Dynamic, Manual

Initial State: Users have completed the previous test. PyCards is now running, with a game in progress

Input/Condition: A group of users are asked to play the game and rate it based ease of use using a scale from 1-5 where 5 is very user friendly

Output/Result: The majority of users give the game an average rating above 3 and greater than or equal to the rating given in the previous test

How test will be performed: A group of randomly chosen people, of different ages and unspecified level of technical expertise will be asked the questions defined in the Appendix and to give answers from a scale from 1-5. The scale maps as: 1 - Very hard to use, 2 - Hard to use, 3 - Some effort to use, 4 - Easy to use, 5 - Very easy to use

3.2.3 Product Integrity

Resource Loading

1. RL1

Type: Structural, Dynamic, Manual

Initial State: The 'cardsets' directory is missing or corrupt

Input/Condition: When loading the application prompts users to either re-download the application or download specifically the cardsets directory

Output/Result: The majority of users are able to find the repository for the program and re-download the executable or the cardset directory

How test will be performed: A group of users selected and willing to participate in this test scenario will be provided with a modified executable or the program's source excluding the cardsets directory. Upon launching the application will attempt to load the directory and find it missing. The application must not crash but instead prompt the user to redownload it

2. RL2

Type: Structural, Dynamic, Manual

Initial State: The 'tiles' directory is missing or corrupt

Input/Condition: Application attempts to load images from the 'tiles' directory

Output/Result: An exception is thrown and handled by using a solid color tile as the window background.

How test will be performed: Before launching the application the 'tiles' directory will be removed from the host system. Success for this test is defined by the application drawing a solid color background and continuing to function normally in all other regards

3.2.4 Security

System Permissions

1. SP1

Type: Structural, Dynamic, Manual

Initial State: Application is located on the host machine's file system

Input/Condition: User launches the application

Output/Result: The application should launch without asking for administrative (root) privileges.

How test will be performed: On Ubuntu, the application will be launched via terminal. It must launch without asking for superuser (root) permissions

On Windows, the application should launch successfully when double-clicked, not requiring the user to right-click and select 'Run as administrator'

Tolerance: OS X implements a functionality called Gatekeeper that restricts the user from opening applications from unidentified developers. This is a security protocol and our app will not attempt to circumvent it. Instead, the user should be able to control-click the app to launch it, after confirming their intention to bypass Gatekeeper

4 Tests for Proof of Concept

4.1 Game Logic

Card Selection

1. PC1

Type: Structural, Dynamic, Automated

Initial State: Klondike game is loaded

Input: User selects card with mouse click

Output: Clicked card is highlighted and tracks mouse cursor if valid selection. If invalid selection the click is ignored. Criteria for valid selection is defined in the SRS

How test will be performed: A random sequence of cards will be programmatically clicked to simulate user clicks. The criteria for valid selection will be evaluated and the invocation of the correct method will be the success criteria for this test

2. PC2

Type: Structural, Static, Manual

Initial State: The rules of the game written using first-order logic

Input: A sequence of possible executions that covers the different conditions

Output: Truth value whether the result of the execution conforms to the rules of the game

How test will be performed: The logic of the game will be mapped as a set of states and transitions, in a control flow diagram. Each possible execution inputted represents a different path. The truth value of the execution is whether or not the rules of the game accept the trace

5 Comparison to Existing Implementation

A subset of our tests compares the functionality of our program to that of the existing implementation. These tests include :

- test PC1 in Tests for Proof of Concept
- test PC2 in Tests for Proof of Concept
- test UN2 in Nonfunctional Requirements - Usability
- test UP2 in Nonfunctional Requirements - Usability

Tests PC1 and PC2 assert that the functionality of PyCards is analogous to the functionality of the existing implementation in terms of enforcing game rules

Tests UN2 and UP2 asserts that the design of PyCards is just as, if not more, user-friendly than that of the existing implementation

6 Unit Testing Plan

The unittest.py module will be used for automated unit testing for this project and the coverage.py module will be used to analyze code coverage

6.1 Unit testing of internal functions

Unit tests are used to validate the internal components such as modules, methods, and functions. To verify the correct behaviour of methods we define the preconditions and postconditions using predicate logic and map the conditions to boolean expressions in our test cases.

More specifically, unit tests were conducted (using unittest.py) for the create() method in each of the game classes as follows:

1. Initial state: Construct new instance of game class
2. Precondition: The game has no previously created stacks
ie. $\text{size}(\text{stacks}) = 0$
 - This condition is necessary and sufficient to test the initial state
 - The ‘create’ method functions analogous to a constructor in that it should only be called using a newly created instance
 - The ‘create’ method should correctly define the initial state of a game and upon termination should leave the game instance in a ready to use (not to mention consistent) state
3. Postcondition: A correctly defined initial state for each and all of the stacks of the game. See unittests for conditions

6.2 Code Coverage Metrics

The coverage.py python module can measure and report code coverage in a number of ways. We will be testing coverage dynamically by running the application (and having the coverage.py module analyze statement coverage). Upon termination, the coverage.py module will then be used to provide a quantifiable coverage report in HTML (for convenience) format. Our application is a game, and as such dynamic code coverage will be measured by running the application and interacting (ie. playing, clicking menus, etc.) such that the application reaches each of the possible states. This includes:

- creating new games of each type
- saving a game in progress
- loading a previously saved game
- attempting to load a game from an invalid file
- performing a canonical set of legal moves for the game in progress
- attempting to perform a canonical set of moves illegal for the game

Coverage of static exception-handling code such as missing directories upon initial execution, invalid configuration files, and runtime exceptions that accounted for (for safety) but never thrown can be tested separately but mostly fall more broadly under the category of system-testing.

7 Appendix

7.1 Symbolic Parameters

- `COMMAND_KEYS` = {'N', 'P', 'U', 'R', 'D', 'H', 'F1', 'F2', `CTRL+N`, `CTRL+P`, `CTRL+Q`}
- `BOUND_ACTIONS` = {newgame, pause, undo, redo, deal, help, quit}
- `MAX_FIND_TIME` = 90 seconds
- `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

- How would you rate the user interface?
- How user-friendly were the in-game dialogs?
- How would you rate the overall game experience?