Faculty of Engineering and Applied Science

SOFE 3980U Software Quality

Winter 2022

Dr. Akramul Azim

TAs: Md Asif Khan, Rezwana Mamata

Assignment 1: Choosing the Right Software Process and Ensuring Quality with Test Automation

Jessica Leishman

(100747155)

March 3, 2022

Table of Contents

GITHUB REPOSITORY:

https://github.com/jessica-leishman/high-rollers

Software Process

The software process used during the development of High Rollers was the agile model, due to the short development time frame and rapidly changing requirements as the developer further explored the capabilities of pygame. Additionally, due to the short development time allotted the developer's previous experience with the Agile process model further cements this choice.

First, requirements were determined during an initial brainstorming session using a whiteboard. Some basic game design/flow was identified through screen mockups. The requirements for the game were transitioned to the GitHub Projects feature for the repository, and some additional functional requirements were identified as per the project guidelines.

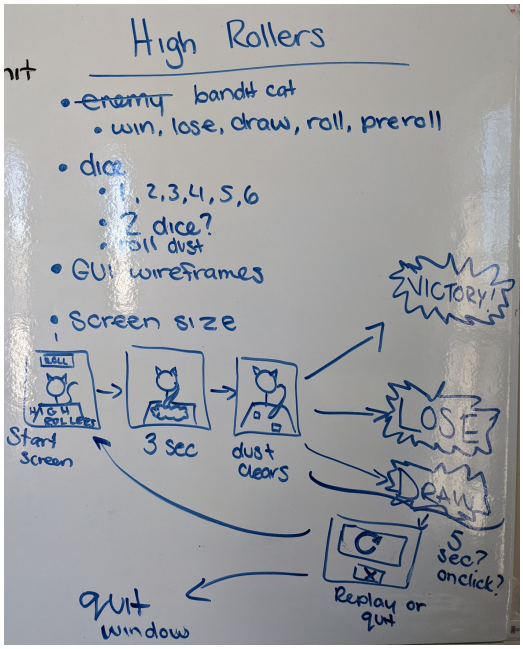

Figure 1: Initial gameflow diagram and assets needed to begin development.

At this point the agile process transitioned into the design phase, and assets were drawn by hand for use. Basic functions were outlined, such as dice rolling, screen navigation menus, simple buttons, and for placing text on the user's screen.

During the development, these initial functions were completed to complete the tasks in the most basic form.

A kanban board was utilised on the GitHub to manage the remaining requirements for further iterations, including adding sound, and score.

GITHUB REPOSITORY: https://github.com/jessica-leishman/high-rollers

Functionality of Software and Code

“High Rollers” is a simple dice rolling game, in which the objective is to roll a higher number than your opponent using a six-sided die. The game was developed using pygame, and all of the assets were hand drawn. This was done as displaying a series of cohesive still images provided the most efficient illusion of movement from frame to frame. The sound effects were obtained from open source soundboards (https://www.101soundboards.com, https://freesound.org).

The mouse position is tracked within the program using pygame.mouse.get_pos().

Upon clicking the “play” button on the first screen (main_menu()), this causes the game to transition to the gameTime() function, where the opponent is loaded in with a blue die. From this point on, a score is present in the top right corner. The game looks for screen transitions by monitoring the pygame event queue for clicks, or exit/quit requests.

Pressing the roll button triggers a custom pygame event to be placed in the event queue after 2 seconds, after which the dust on the table will clear. At this point, the screen will transition to either a win, lose, or draw screen, and the die assets corresponding to the numbers generated using secrets.randbelow(5)+1 are displayed. The score will be incremented, decremented, or will remain the same depending on the outcome of the round. Two buttons prompt the user if they wish to play again, or quit the application. The score will continue to be updated from round-to-round until the “High Rollers" game is closed, either using the [X] button for the game window, or one of the quit buttons within the game’s interface.

How to Run

1. ‘pip install pygame’
2. ‘pip install -U matplotlib’
3. Run hrDriver.py through an IDE, or through the command line with ‘python hrDriver.py’. If you are having issues with the command line, try an IDE.
4. Play game :) (ensure volume is on!)

Ensure all game files, assets, and sounds are stored within the same directory, otherwise they will not load correctly.

For running the test suite: ‘pip install -U pytest’. Then, ‘pytest testsuite.py’ from within the game’s directory.

Unfortunately, pygame games cannot be packaged as an executable file. Setup must be done manually as outlined above.

GITHUB REPOSITORY: https://github.com/jessica-leishman/high-rollers

Gameplay Screenshots

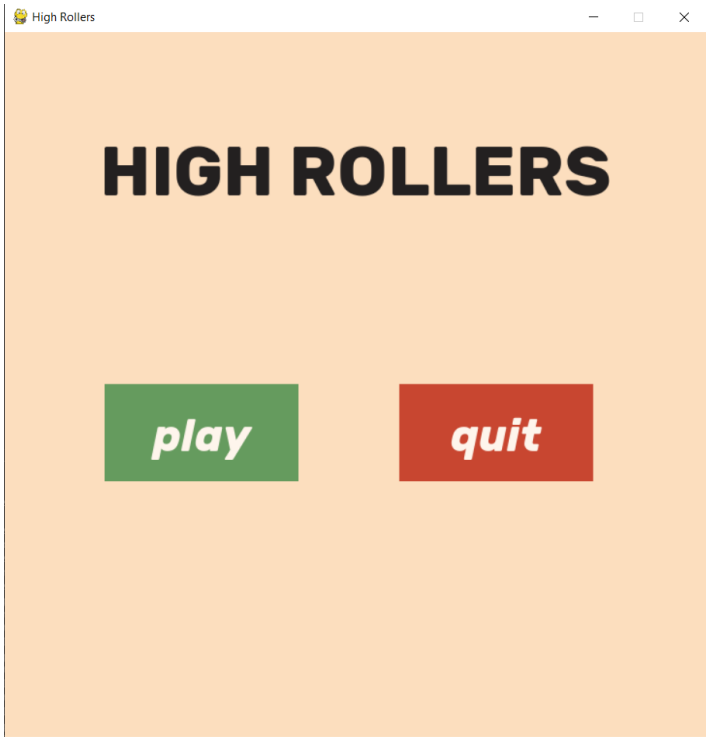

Figure 2: Main menu upon game launch, with announcer sound "Let's Play a Game!"

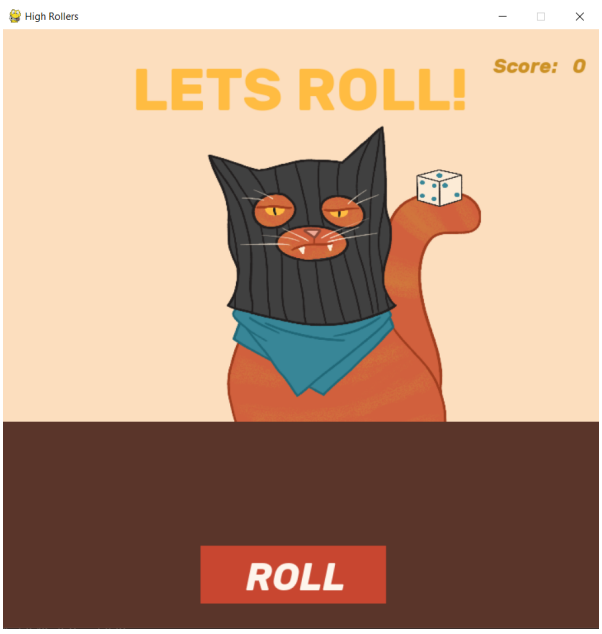

Figure 3.1: Initial gameplay screen, with score 0

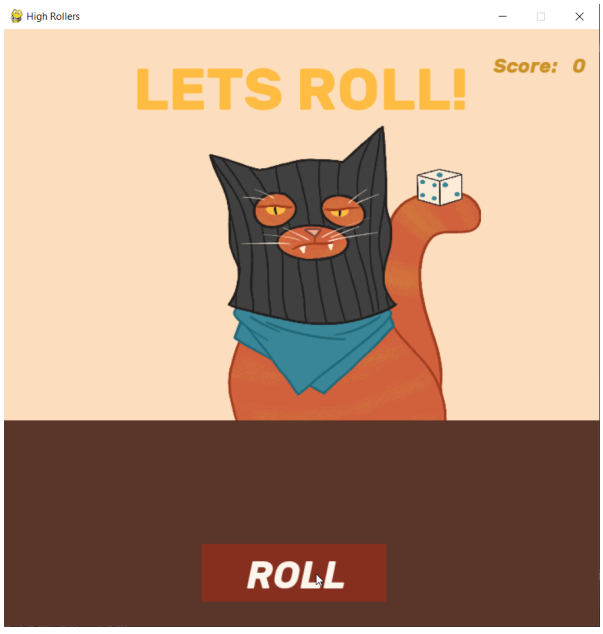

Figure 3.2: Hover effect on gameplay screen button.

GITHUB REPOSITORY: https://github.com/jessica-leishman/high-rollers

Figure 4: Dust cloud while random dice numbers are generated. Dice rolling sound effect played

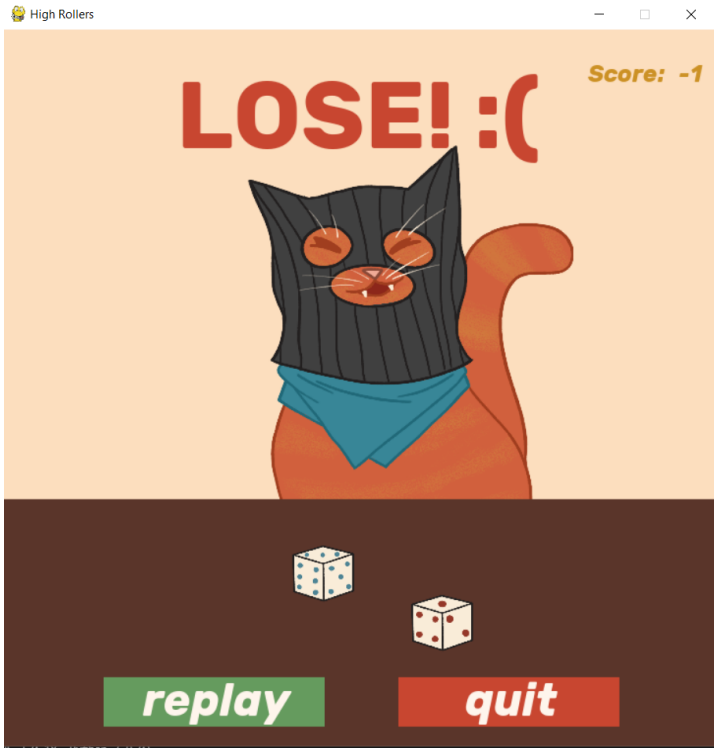

Figure 5: User loss screen, with greater value on blue (computer) die. Button hover effect not pictured.

GITHUB REPOSITORY: https://github.com/jessica-leishman/high-rollers

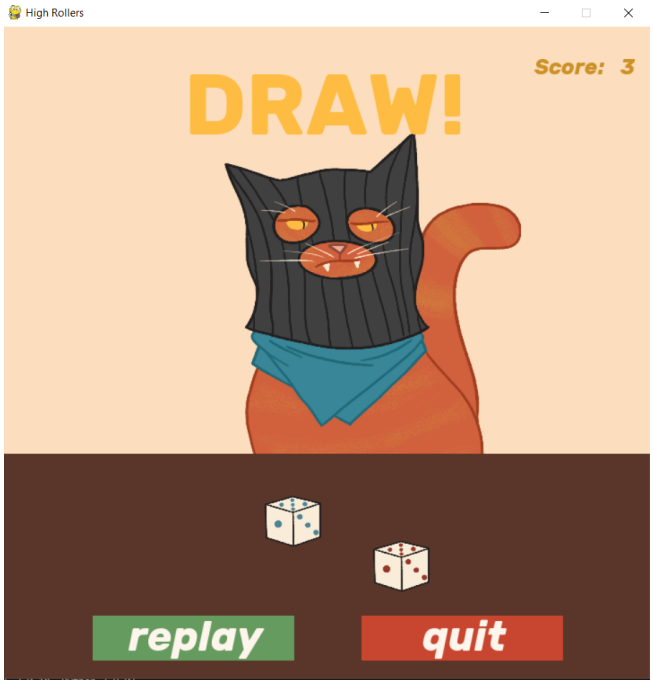

Figure 6: Draw result screen, with equal values on both dice. Button hover effect not pictured. Cat hissing/cry sound effect to show

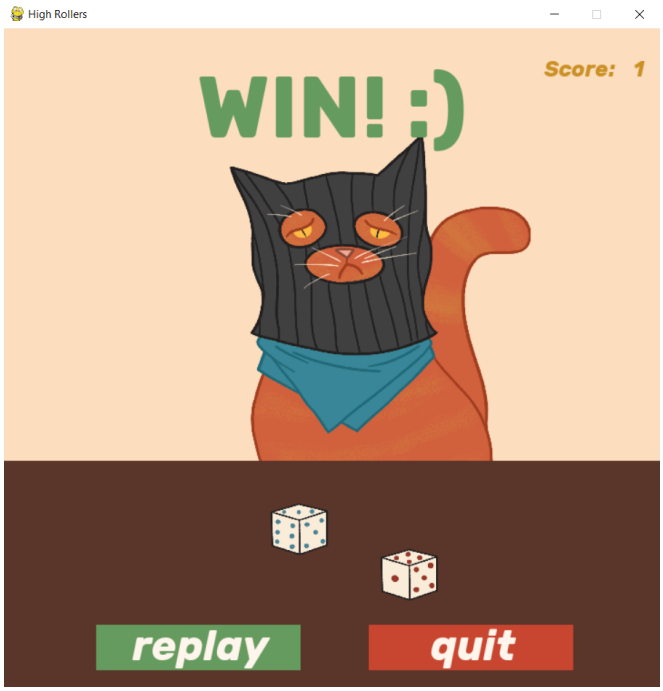

Figure 7: User victory screen, with greater value on red (user) die. Button hover effect not pictured. Victory fanfare sound effect.

GITHUB REPOSITORY: https://github.com/jessica-leishman/high-rollers

Test Automation

The test cases developed are contained within testsuite.py. These test cases were developed for use with pytest. Pytest provides a straightforward testing platform, that allows for easy execution of unit tests. In order to use pytest on your host system, use 'pip install -U pytest'.

The results of these tests are included below:

```
C:\Users\justa\Documents\GitHub\high-rollers>coverage run -m pytest -v testsuite.py && coverage report -m
============================================= test session starts =============================================
platform win32 -- Python 3.9.0, pytest-7.0.1, pluggy-1.0.0 -- c:\users\justa\appdata\local\programs\python\python39\python.exe
cachedir: .pytest_cache
rootdir: C:\Users\justa\Documents\GitHub\high-rollers
plugins: anyio-3.1.0, mock-3.7.0
collected 25 items

testsuite.py::testScreenExist PASSED                                                                  [  4%]
testsuite.py::testWidthHeight PASSED                                                                  [  8%]
testsuite.py::testWinDisp PASSED                                                                      [ 12%]
testsuite.py::testLoseDisp PASSED                                                                     [ 16%]
testsuite.py::testDrawDisp PASSED                                                                     [ 20%]
testsuite.py::testDieSet PASSED                                                                       [ 24%]
testsuite.py::testDispDie1 PASSED                                                                     [ 28%]
testsuite.py::testDispDie2 PASSED                                                                     [ 32%]
testsuite.py::testDispDie3 PASSED                                                                     [ 36%]
testsuite.py::testDispDie4 PASSED                                                                     [ 40%]
testsuite.py::testDispDie5 PASSED                                                                     [ 44%]
testsuite.py::testDispDie6 PASSED                                                                     [ 48%]
testsuite.py::testWinCat PASSED                                                                       [ 52%]
testsuite.py::testLoseCat PASSED                                                                      [ 56%]
testsuite.py::testDrawCat PASSED                                                                      [ 60%]
testsuite.py::testPlayGame PASSED                                                                     [ 64%]
testsuite.py::testExit PASSED                                                                         [ 68%]
testsuite.py::testStartRoll PASSED                                                                    [ 72%]
testsuite.py::testPlayAgain PASSED                                                                    [ 76%]
testsuite.py::testQuit PASSED                                                                         [ 80%]
testsuite.py::testScoreUp PASSED                                                                      [ 84%]
testsuite.py::testScoreDown PASSED                                                                    [ 88%]
testsuite.py::testScoreDraw PASSED                                                                    [ 92%]
testsuite.py::testPalette PASSED                                                                      [ 96%]
testsuite.py::testTable PASSED                                                                        [100%]

============================================== 25 passed in 1.09s ==============================================
Name             Stmts   Miss  Cover   Missing
------------------------------------------------
highrollers.py     312    234    25%   54-57, 62-114, 147-149, 154-198, 247-253, 258-308, 313-380, 385-450, 455-517
testsuite.py       106      0   100%
------------------------------------------------
TOTAL              418    234    44%

C:\Users\justa\Documents\GitHub\high-rollers>
```

Figure 8: Detailed coverage report of pytest testsuite.py.

```
C:\Users\justa\Documents\GitHub\high-rollers>pytest testsuite.py
============================================================================= test session starts =============================================================================
platform win32 -- Python 3.9.0, pytest-7.0.1, pluggy-1.0.0
rootdir: C:\Users\justa\Documents\GitHub\high-rollers
plugins: anyio-3.1.0, mock-3.7.0
collected 25 items

testsuite.py .........................                                                                                                                                   [100%]

============================================================================== 25 passed in 0.80s ==============================================================================
```

Figure 9: pytest execution of testsuite.py

The tests administered were developed to verify the major functions of the game logic. This was difficult to do as pygame elements were incompatible with some aspects of the testing, and thus the code had to be altered to meet the test purpose outlined during the brainstorming phase of an agile iteration. As a result of this, the pygame event queue monitoring loop used on all screens of the game could not be tested, however, an alternative test type was developed. In

GITHUB REPOSITORY: https://github.com/jessica-leishman/high-rollers

the modified test type the buttons are tested for collision, to ensure that clicks will be detected in the appropriate region to initiate the corresponding next screen function.

A description of each test can be found in the table below:

Test Case Name	Description
testScreenExist	Tests the resolution for the screen window is set.
testWidthHeight	Test the width and height are set from the resolution.
testWinDisp	Test the dice passed result in a "win" status detection.
testLoseDisp	Test the dice passed result in a “lose” status detection.
testDrawDisp	Test the dice passed result in a “draw” status detection.
testDieSet	Tests the random function used to role sets a value to the die.
testDispDie1	Tests display for red (user) and blue (opponent) dice assets “1”.
testDispDie2	Tests display for red and blue dice assets showing “2”.
testDispDie3	Tests display for red and blue dice assets showing “3”.
testDispDie4	Tests display for red and blue dice assets showing “4”.
testDispDie5	Tests display for red and blue dice assets showing “5”.
testDispDie6	Tests display for red and blue dice assets showing “6”.
testWinCat	Tests that catPath function obtains the correct image for winner state = 0. (User loss)
testLoseCat	catPath function obtains the correct image for winner state = 1. (Computer loss)
testDrawCat	catPath function obtains the correct image for winner state = 2. (Draw)
testPlayGame	Test collision with the "PLAY" button on the main screen is detected.
testExit	Test collision with the "QUIT" button on the main screen is detected.
testStartRoll	Test collision with "ROLL" button triggers
testPlayAgain	Test collision with play again button (after a roll)
testQuit	Test collision with play again button (after a roll)
testScoreUp	Test that increasing the score using the updateScore method

GITHUB REPOSITORY: https://github.com/jessica-leishman/high-rollers

	with the winner state 1 (victory) causes it to increase
testScoreDown	Test that increasing the score using the updateScore method with the winner state 0 causes it to decrease.
testScoreDraw	Test that increasing the score using the updateScore method with the winner state 2 causes no change
testPalette	Test that all palette colours are defined
testTable	Test that the table generation function is called and displays

Lines missing from coverage and how they were instead tested:

Lines	Description	Test Case Name
54-57	Renders text object passed to function along with message	n/a. Could not test.
62-114	Main_menu game loop. Pygame buttons, tested with collision instead. Pygame event loop not tested.	testPlayGame(), testExit()
147-149	Displays game score on screen	n/a, uses lines 53-56
154-198	gameTime game loop. Pygame button, tested with collision instead. Pygame event loop not tested.	testStartRoll()
247-257	showDice() function, displays the dice images selected onto the screen using pygame display	n/a. Could not test, pygame.
258-308	gameLogic game loop, with dust_clear_event. Pygame event loop, not tested.	n/a. Could not test.
313-380	Pygame sound, and pygame event loop for victory. Pygame buttons tested with collision instead.	testPlayAgain(), testQuit()
385-450	Pygame sound, and pygame event loop for loss. Pygame buttons tested with collision instead.	testPlayAgain(), testQuit()
455-517	Pygame sound, and pygame event loop for draw. Pygame buttons tested with collision instead.	testPlayAgain(), testQuit()

GITHUB REPOSITORY: https://github.com/jessica-leishman/high-rollers

Challenges Faced During Development and Test Automation

Development of the game was fairly smooth once assets were created, and some time was spent becoming familiar with the pygame documentation (available at: https://www.pygame.org/docs/ref/transform.html, down at time of writing due to Ukraine-Russian relations). The most time consuming process was developing a game idea with a reasonable scope for the short time period allotted for development.

Pygame elements could not be used within pytest assert statements, which required the functions to modify the way in which data was obtained from game logic values.

To attempt to achieve higher test coverage, the mock plugin was explored. The purpose of the mock plugin is to “mock” or mimic user input without the need for it to occur in real time. This allows a variety of input combinations and related functions to be tested and verified through automated testing.

Unfortunately, mock was not compatible with pygame elements or the pygame event queue and thus the event queue for button presses and mouse clicks could not be simulated for testing. As stated previously in the *Test Automation* section, an alternative test type was instead used to check the collision of buttons with the x and y coordinates of the mouse, which results in a higher coverage that the coverage report of 25% indicates.

GITHUB REPOSITORY: https://github.com/jessica-leishman/high-rollers