McAlvain Colin

Keith Erin

CS333, Testing & DevOps

2023/04/25

Final Project Design Documentation: Uno Game by “bennuttall”

Introduction:

This paper will discuss the testing and automation of building processes for a digital version of a UNO game simulator software, written primarily in Python by a user with the username "bennuttall." The GitHub repository link for the software is provided in the citations. The main objective of this project is to automate the software's building and deployment processes, while also utilizing researched technology with an explanation of why specific technology was chosen. Additionally, the software will undergo a minimum of 75% unit testing, and at least five integration tests will be included in the project.

Software Project Selection:

The choose code is written in Python programming language and uses the UnoGame class from the "uno" module to simulate a game of Uno.

The program generates a random number of players between 2 and 15 and creates a new instance of the UnoGame class with that number of players.

The program loops while the game is still active according to the rules defined by the wiki-site. Within the loop, the code keeps track of the number of turns played and identifies the current player. If the current player can play a card, the code checks if any of the cards in the player's hand are playable on the current card. If so, it selects the first playable card, and if it is a black card, it randomly selects a color for the card to be played. The code then prints a message indicating the card played by the player and updates the game accordingly by playing the selected card. If the current player cannot play any card, the code prints a message indicating that the player is picking up a card and updates the game by playing a null card.

The loop continues until the game is no longer active. Once the game is over, the code prints a message indicating the number of players in the game and the number of cards played during the game.

Classes:

- UnoGame: This class contains the necessary attributes and methods to simulate a game of Uno. It includes methods for shuffling and dealing cards, playing cards, and checking game status.
- UnoPlayer: This class represents a player in the game and has properties like player ID, hand, and methods like checking if the player has any playable cards.
- UnoCard: This class represents a single card in the game and has properties like color, card type, color short, card type short, and methods like a check for valid card, check if card is playable.
- ReversibleCycle: This class represents an interface to an iteratoratable list which can be infinitely cycled and reversed. It includes a method for next, and attributes for position and direction.
- AIUnoGame: This class represents a full Uno game that includes an UnoGame, players, hand, and choices. It includes methods to run through a full game of Uno and lets to user play with the system.

The code imports Python's random and itertools libraries. It defines some constants like the list of colors, all colors, numbers, special card types, color card types, black card types, and card types. Then it defines the UnoCard class, which has methods to validate and represent the card. It also defines the UnoPlayer class, which has a method to check if the player has any playable cards. Finally, it defines the UnoGame class, which represents the game and has methods like dealing the cards, creating the deck, iterating over the players, and checking if a player has won the game.

NOTE: This github repo has a testing file included from the original author. These files will not be used for the CS333 final project but will be turned in with the rest of the program for transparency. This test code will not be used for this project. All unit tests in provided in reference to this project will be written by the authors of this paper and will be original to this project. Correlating tests may exist due to the nature of the testing process.

Overall, this code randomly generates a game of Uno with a random number of players and plays through the game until it is complete.

Functionality and Technologies:

Functionality:

1. This code functions as a digital Uno game.
2. This code can simulate 2-15 players.
3. This code has self-play functionality with computer generated game play for NPC.
4. This code simulates a deck of cards that mirrors the uno deck.
    a. If this deck runs out of cards it is shuffled and put back into play.
    b. Cards in play are not regenerated.
5. This code has a graphic user interface with visible cards that have images of official Uno cards with a black back.

Technologies:

1. Libraries Used
    a. Shuffle
    b. Choice
    c. Product
    d. Repeat
    e. Chain
    f. Unittest
    g. Tread
    h. Sleep
    i. Random
    j. itertools
2. Outside Software
    a. Github
    b. Python Unittest
    c. Python3
    d. GIthub
    e. Docker

Testing Plan:

The testing outline for the UnoGame class using the unittest framework covers the following scenarios:

1. Test that the UnoGame class is initialized with the correct number of players.
2. Test that the UnoGame class initializes the deck correctly.
3. Test that the UnoGame class deals the cards to the players correctly.
4. Test that the play method of the UnoGame class plays a card correctly.
5. Test that the play method of the UnoGame class picks up a card correctly if the player cannot play.
6. Test that the play method of the UnoGame class handles the skip card correctly.
7. Test that the play method of the UnoGame class handles the reverse card correctly.
8. Test that the play method of the UnoGame class handles the draw two card correctly.
9. Test that the play method of the UnoGame class handles the wild card correctly.
10. Test that the play method of the UnoGame class handles the wild draw four card correctly.

The testing outline for the UnoCard class using the unittest framework covers the following scenarios:

1. Test that the UnoCard class is initialized with the correct color.

2. Test that the UnoCard class is initialized with the correct type.
3. Test that the UnoCard class can catch a bad initialization of color.
4. Test that the UnoCard class can catch a bad initialization of type.
5. Test that the UnoCard class is initialized with the reverse.
6. Test that the UnoCard class is initialized with the correct name.
7. Test that the UnoCard class is matches a card of the same color.
8. Test that the UnoCard class is matches a card of the same value.
9. Test that the UnoCard class is does not matches a card of a different color.
10. Test that the UnoCard class is does not matches a card of a different value.
11. Test that the UnoCard class is playable against another card.
12. Test that the UnoCard class is not playable against another card.

The testing outline for the UnoPlayer class using the unittest framework covers the following scenarios:

1. Test that the UnoPlayer class is initialized with the correct cards.
2. Test that the UnoPlayer class is initialized with the correct player_id.
3. Test that the UnoPlayer class represents the player id correctly.
4. Test that the UnoPlayer class can play when the player has a valid UnoCard.
5. Test that the UnoPlayer class cannot play when the player has an invalid UnoCard.

The testing outline for the ReversibleCycle class using the unittest framework covers the following scenarios:

1. Test that the ReversibleCycle class is initialized with the correct iterable list.
2. Test that the ReversibleCycle class is initialized with the correct position.
3. Test that the ReversibleCycle class is initialized with the correct direction of play.
4. Test that the ReversibleCycle class position iterates correctly after a valid card play.
5. Test that the ReversibleCycle class position iterates correctly after an invalid card play.
6. Test that the ReversibleCycle class reverses play direction correctly when reverse is used.

The testing outline for the AIUnoGame class using the unittest framework covers the following scenarios:

1. Test that the AIUnoGame class is initialized with the correct game.
2. Test that the AIUnoGame class is initialized with the correct players.
3. Test that the AIUnoGame class is initialized with the correct player index.
4. Test that the AIUnoGame class is prints the correct player hand.
5. Test that the AIUnoGame class next has the correct player id for each turn.
6. Test that the AIUnoGame class next holds the correct current card in play.
7. Test that the AIUnoGame class prints the current players hand each turn.

For the integration tests, the following five scenarios are tested:

1. Test that the game ends when a player has no cards left.
2. Test that the game correctly handles the case when the draw pile runs out of cards.
3. Test that the game correctly handles the case when a player tries to play an invalid card.
4. Test that the game correctly handles the case when a player tries to play a card that does not match the color or value of the current card.
5. Test that the game correctly handles the case when a player tries to play a wild or wild draw four card and sets the new color to an invalid value.

Overall, these tests cover a significant portion of the code and test the various functionalities of the UnoGame class in different scenarios.

Source Code Centralization:

This project will use Github for source code centralization, implementing a master, dev, and feature branch approach, to segment workflow processes.

GitHub is a good software choice for source code centralization in software design for several reasons. Firstly, it provides a centralized location for developers to store and manage their code. This makes it easy to collaborate and share code with others. Secondly, GitHub offers robust version control features, which allow developers to track changes to their code over time. This can help to prevent errors and conflicts when multiple developers are working on the same project. Additionally, GitHub offers a range of tools for code review, issue tracking, and project management, making it a comprehensive platform for software development. Finally, GitHub is widely used in the industry, which means that developers who are familiar with the platform will be well-equipped to work on projects with other teams and companies. Overall, GitHub's features and popularity make it an excellent choice for source code centralization in software design.

Build and Deployment Automation:

This project will use Docker for automation of building and testing. Docker is a good software choice for build and development automation of software design due to its ability to package an application and its dependencies into a self-contained, portable container. This containerization approach ensures that the application will run reliably and consistently across different environments, which is crucial for effective build and development automation.

Docker also offers several benefits, including efficient resource utilization, faster deployment, and increased scalability. By using Docker, developers can easily create, test, and deploy applications in a highly automated and streamlined manner, reducing the time and effort required to manage complex software environments. Furthermore, Docker's open-source

community provides a wealth of resources, tools, and plugins to enhance the development process further.

Overall, Docker's containerization approach, portability, and community support make it an excellent software choice for build and development automation, enabling developers to focus on creating high-quality software with greater speed and efficiency.

Conclusion:

In conclusion, this project is aimed at automating the building and deployment processes of the Uno game simulator software, developed by "bennuttall," while incorporating researched technology. The code was primarily written in Python and utilized the UnoGame class from the "uno" module to simulate a game of Uno with 2-15 players. The program randomly generated a game, keeping track of the number of turns played and identifying the current player, playing cards, and checking the game's status until the game was complete. The code includes various classes, including UnoGame, UnoPlayer, UnoCard, ReversibleCycle, and AIUnoGame. The testing plan included unit testing and integration testing to test the code's functionality, and the unittest framework will be utilized to cover various scenarios. The software was tested for functionalities like dealing the cards, playing cards correctly, picking up the cards, handling the skip card, handling the reverse card, and more. The software will also incorporate outside software like Github, Docker, and Python Unittest. The successful implementation of automated building and deployment processes, along with the utilization of researched technology and the thorough-testing, makes this software an strong digital version of the Uno game simulator.

Citations:

Bennuttall. (2017, August 31). *Bennuttall/Uno: Python implementation of the card game uno*. GitHub. Retrieved April 25, 2023, from https://github.com/bennuttall/uno

Wikimedia Foundation. (2023, March 28). *Uno (card game)*. Wikipedia. Retrieved April 25, 2023, from https://en.wikipedia.org/wiki/Uno_(card_game)#Official_rules