Final Project: Hangman
Computing Fundamentals
EECE2140

Michael Brodskiy
Brodskiy.M@Northeastern.edu

April 28, 2023

Instructor: Professor Fanaei

1 Introduction

The purpose of the final project in this course is to unify concepts covered throughout the entirety of the course; as such, this final project recreates the popular hangman game. There are two modes, single and multi player. When single player is selected, a word is selected at random (using the `random` module) from a word bank. The player then has two options: guessing a letter or guessing the phrase, both of which work as one would expect. Every time a letter is guessed, it is added to the list of guessed letters, which stop the player from guessing the same letter twice. The guesser has 5 attempts to guess correctly until they are "hung". The difference between single and multi player is that, with multiplayer, a custom phrase may be entered. Once a game ends, the user is prompted whether they want to play again, and the process is restarted.

On the technical side of things, two classes are implemented in the code: `hangman` and `game`. The `game` class inherits from the `hangman` class, which is essentially just a class to display the text-based hangman figure. The `game` class contains several functions, each of which is used to implement a certain aspect of the game, such as `end()`, which checks whether the game should end, `enter_phrase`, which handles custom phrase input, and `generate_secret`, which converts an entered phrase to underscores and spaces. Together with the code to create a `game` object, the game may be easily implemented and played.

2 Components

First and foremost, there is the `hangman` class, which is essentially just used to create a hangman figure. Within it, there is an `__init__()` function, which is not very important, and an `__str__()` function, which prints the figure.
The `game` class inherits the hangman class, but has much more functionality in terms of running the actual game. The functions it contains are as follows:

- `__init__()` — Initializes the superclass, and creates `mode`, `limbs_lost`, `totalguesses`, and `guessedLetters` class attributes.
- `mode` getters and setters — Used to verify that `mode` is always either 0 or 1
- `enter_word()` — Executes the `generate_secret()` function, and stores the word in the `word` class attribute
- `guess_part()` — Used to guess a letter
  - If more than a single letter is entered, causes exits function
  - Adds the guessed letter to the `guessedLetters` list, if not there already; if it is already there, exits the function
  - Checks for the index of the guess; catches a ValueError by setting `index` to 1 instead; if the index is not -1, each instance of a letter is replaced, and the `totalguesses` attribute is increased by one; if the index is -1 (not found), the player loses a limb (`limbs_lost` is increased by 1), and `totalguesses` is increased

- `guess_phrase()` — Used to check whether the user has entered the correct phrase; if yes, the game ends and user wins; if not, the user loses a limb and the game continues.
- `generate_word()` — Uses the `random` module to randomly select a word from a word bank (for single player mode)
- `generate_secret()` — Converts an entered word to underscores for the letters and keeps spaces as spaces
- `end()` — Also implements exception handling to determine whether the game has ended (all limbs lost or all letters guessed)
- `menu()` — Used to print a menu to ask whether the user would like to enter a letter or phrase
- `play()` — Implements all of the above functionality, in addition to the `__str__()` function to run the actual game
- `__str__()` — Uses the string function from `hangman`, in addition to class attributes, to print a tui game window

The main code simply contains some code to request a mode selection by the user, create a `game` object, and, upon game end, asks whether the user would like to play again (using a `while` loop)

3 Running the Code

Given that this project is contained within a single file, the only necessity is to have the `random` module installed; the project may be run with the following command:

```
>_ python3 hangman.py
```

4 External Libraries

The only external library used was the `random` module

5 Conclusion

The implementation of this program, most importantly, solidified two concepts for me: the idea of object-oriented programming in python, and the use of exception handling. It took me some time to figure out why simply using `<something>.index(<text>) == -1` did not work, and how to fix it. Additionally, using protected classes and inheritance allowed me to get a better feel for abstraction, and the fabrication of an input-based `__str__()` function allowed for a greater understanding of polymorphism.

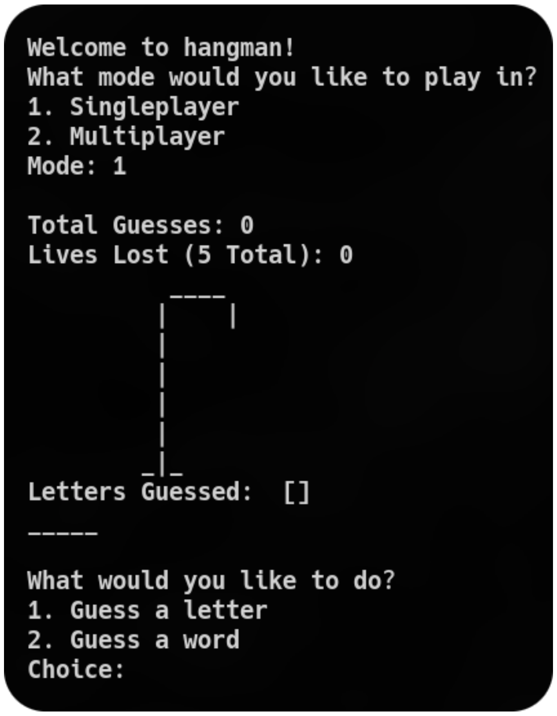
6 Appendix

Listing 1: Full Program Code

```python
"""
 * ============================================================
 *
 *        Filename:   hangman.py
 *        Assignment: EECE 2140 Final Project
 *        Title: Hangman
 *
 *     Description:  A rendition of the classic hangman game
 *
 *         Version:   1.0
 *         Created:   04/20/2023
 *        Revision:   N/A
 *          Python:   Python 3.9.2
 *
 *          Author:   M.Brodskiy
 *
 * ============================================================
"""

import random

class hangman: # Create a class for the game

    def __init__(self): # Initialize the class

        # Create a tui-based figure for the game
        self.figure = ["\t  ____\t\n\t |    |\n\t |    \n\t |   \n\t |    \n\t |   \n\t_|_", "\t  ____\t\n\t |    |\n\t |    o\n\t |   \n\t |    \n\t |   \n\t_|_", "\t  ____\t\n\t |    |\n\t |    o\n\t |   /|\n\t |    \n\t |   \n\t_|_", "\t  ____\t\n\t |    |\n\t |    o\n\t |   /|\\\n\t |    \n\t |   \n\t_|_", "\t  ____\t\n\t |    |\n\t |    o\n\t |   /|\\\n\t |    |\n\t |   / \n\t_|_", "\t  ____\t\n\t |    |\n\t |    o\n\t |   /|\\\n\t |    |\n\t |   / \\\n\t_|_"]

    def __str__(self, lost_limbs):

        # Print the figure above
        return self.figure[lost_limbs]

class game(hangman):

    def __init__(self, mode = 0): # Initialize

        super().__init__() # Initialize the superclass

        # Create a value to keep track of the mode,
            protected
        self._mode = mode # 0 for single player, 1 for
            multi player
        self.limbs_lost = 0 # Create a lives lost variable
        self.totalguesses = 0 # Create a total guess value
        self.guessedLetters = [] # Create guessed letter
            bank


    @property # read property of the mode
    def mode(self):
        return self._mode

    @mode.setter # write property of the mode
    def mode(self, mode):
        if mode not in [0, 1]: # Raise an error if mode not
             valid
            raise ValueError("The mode has to be one or
                zero")
        else:
            self._mode = mode

    def enter_word(self, word):

        self.generate_secret(word) # generates the secret
        self.word = word # sets the word to entered word

    def guess_part(self, guess): # Guess part of the word

        if (len(guess) > 1): # Checks that a single letter
            was entered
            print("Not a letter")
            return -1

        guess = str(guess.lower()) # Make the guess lower
            case
        if (guess not in self.guessedLetters): # Checks if
            a letter was already guessed
            self.guessedLetters.append(guess)
        else:
```

```python
            print("Letter has already been guessed!")
            return -1

        try: # If there is a ValueError, return -1
            index = self.word.index(guess)
        except ValueError:
            index = -1
        if (index == -1):
            self.totalguesses += 1 # Increase total guesses
            self.limbs_lost += 1 # Otherwise, lose a limb
            return -1

        while (index != -1): # Find if guess is in the word

            # Replace secret to contain all instances of guess
            self.secret = self.secret[:self.word.index(guess)] + guess + self.secret[self.word.index(guess) + len(guess):]
            # Replace the word contents of guess to uppercase so replacement is not repeated
            self.word = self.word[:self.word.index(guess)] + guess.upper() + self.word[self.word.index(guess) + len(guess):]

            try: # If there is a ValueError, return -1
                index = self.word.index(guess)
            except ValueError:
                index = -1

        self.totalguesses += 1 # Increase total guesses

        return 0

    def guess_phrase(self, guess): # Guess the whole phrase

        if (self.word.lower() == guess.lower()): # If correct return true
            self.secret = self.word.lower()
            self.totalguesses += 1
            return True
        else: # Otherwise, lose a limb and return false
            self.totalguesses += 1
            self.limbs_lost += 1
            return False

    def generate_word(self): # Used to generate a new word

        # Randomly obtain a word from the word bank below
        bank = ["abruptly", "absurd", "abyss", "affix" "
            askew" "avenue" "awkward" "axiom" "azure" "
            bagpipes" "bandwagon" "banjo" "bayou" "beekeeper
            " "bikini" "blitz" "blizzard" "boggle" "bookworm
            " "boxcar" "boxful" "buckaroo" "buffalo" "
            buffoon" "buxom" "buzzard" "buzzing" "buzzwords"
             "caliph" "cobweb" "cockiness" "croquet" "crypt"
             "curacao" "cycle" "daiquiri" "dirndl" "disavow"
             "dizzying" "duplex" "dwarves" "embezzle" "equip
            " "espionage" "euouae" "exodus" "faking" "
            fishhook" "fixable" "fjord" "flapjack" "flopping
            " "fluffiness" "flyby" "foxglove" "frazzled" "
            frizzled", "fuchsia", "funny", "gabby", "galaxy"
            , "galvanize", "gazebo", "giaour", "gizmo", "
            glowworm", "glyph", "gnarly", "gnostic", "gossip
            ", "grogginess", "haiku", "haphazard", "hyphen",
             "iatrogenic", "icebox", "injury", "ivory", "ivy
            ", "jackpot", "jaundice", "jawbreaker", "jaywalk
            ", "jazziest", "jazzy", "jelly", "jigsaw", "jinx
            ", "jiujitsu", "jockey", "jogging", "joking", "
            jovial", "joyful", "juicy", "jukebox", "jumbo",
            "kayak", "kazoo", "keyhole", "khaki", "kilobyte"
            , "kiosk", "kitsch", "kiwifruit", "klutz", "
            knapsack", "larynx", "lengths", "lucky", "luxury
            ", "lymph", "marquis", "matrix", "megahertz", "
            microwave", "mnemonic", "mystify", "naphtha", "
            nightclub", "nowadays", "numbskull", "nymph", "
            onyx", "ovary", "oxidize", "oxygen", "pajama", "
            peekaboo", "phlegm", "pixel", "pizazz", "
            pneumonia", "polka", "pshaw", "psyche", "puppy",
             "puzzling", "quartz", "queue", "quips", "
            quixotic", "quiz", "quizzes", "quorum", "
            razzmatazz", "rhubarb", "rhythm", "rickshaw", "
            schnapps", "scratch", "shiv", "snazzy", "sphinx"
            , "spritz", "squawk", "staff", "strength", "
            strengths", "stretch", "stronghold", "stymied",
            "subway", "swivel", "syndrome", "thriftless", "
            thumbscrew", "topaz", "transcript", "transgress"
            , "transplant", "triphthong", "twelfth", "
            twelfths", "unknown", "unworthy", "unzip", "
            uptown", "vaporize", "vixen", "vodka", "voodoo",
             "vortex", "voyeurism", "walkway", "waltz", "
            wave", "wavy", "waxy", "wellspring", "wheezy", "
            whiskey", "whizzing", "whomever", "wimpy", "
            witchcraft", "wizard", "woozy", "wristwatch", "
            wyvern", "xylophone", "yachtsman", "yippee", "
            yoked", "youthful", "yummy", "zephyr", "zigzag",
             "zigzagging", "zilch", "zipper", "zodiac", "
            zombie"]

        self.word = bank[random.randint(0, len(bank))]

        return (self.word, self.generate_secret(self.word))

    def generate_secret(self, word): # Used to generate a
        secret corresponding to the word, must be called
        after word generation

        self.secret = ""

        for i in word:

            if (i != " "): # Set all letters to underscores

                self.secret += "_"

            else: # Keep spaces as spaces

                self.secret += " "

        return self.secret

    def end(self):

        # Check for game end
        try: # If there is a ValueError, return −1
            index = self.secret.index("_")
        except ValueError:
            index = −1
        if (index == −1 or self.limbs_lost == 5):
            return True

        return False

    def menu(self): # Prints the menu
        return int(input("What would you like to do?\n1.
            Guess a letter\n2. Guess a word\nChoice: "))

    def play(self): # Runs the game
```

```python

        if (self.mode == 0): # If the mode is single player, generate a word

            self.generate_word()

        else: # If mode is 1, have other player enter a phrase

            self.enter_word(input("enter a word: ").lower())

        while (not self.end()): # Loops until game is won or lost
            self.__str__() # Prints this object as a string
            choice = self.menu() # Prints menu
            if (choice == 1): # Routes to guess letter, phrase, or choose again
                self.guess_part(input("Guess: "))
            elif (choice == 2):
                self.guess_phrase(input("Guess: "))
            else:
                print("Choose again")
        if (self.limbs_lost < 5): # Win state
            print("You guessed the phrase! You win!")
            print(f"Your accuracy was: {((1 - (self.limbs_lost / self.totalguesses)) * 100):.2f}%")
        else: # Loss state
            print(super().__str__(self.limbs_lost))
            print("You were hung! You lose!")
            print(f"The correct phrase was \"{self.word.lower()}\"")

    def __str__(self): # Prints the game
        print() # Spacer
        print("Total Guesses:", self.totalguesses) # Prints guess count
        print("Lives Lost (5 Total):", self.limbs_lost) # Prints Lives
        print(super().__str__(self.limbs_lost)) # Prints hangman figure
        print("Letters Guessed: ", self.guessedLetters) # Print guessed letter list
        print(self.secret) # Prints the remaining characters to guess
        print() # Spacer

# The code below is an example of how the above classes would be implemented
mSel = int(input("Welcome to hangman!\nWhat mode would you like to play in?\n1. Singleplayer\n2. Multiplayer\nMode: ")) - 1 # Obtains a mode
while (mSel not in [0, 1]): # If mode is invalid, tries again
    mSel = int(input("Invalid option, choose again: "))
Game = game(mSel) # Creates object with mode
Game.play() # Plays game
while (mSel != -1): # Asks user to play again
    choice = input("Play again? (y/n): ")[0].lower()
    if (choice == "y"):
        Game = game(int(input("What mode would you like to play in (0 to exit)?\n1. Singleplayer\n2. Multiplayer\nMode: ")) - 1) # Runs mode select again
        Game.play()
        mSel = -1
    else: # If user is done, exits
        print("Exiting...")
        mSel = -1
```



```
Welcome to hangman!
What mode would you like to play in?
1. Singleplayer
2. Multiplayer
Mode: 1

Total Guesses: 0
Lives Lost (5 Total): 0
          ____
         |    |
         |
         |
         |
         |
        _|_
Letters Guessed:  []
-----

What would you like to do?
1. Guess a letter
2. Guess a word
Choice:
```

Figure 1: Launching a Single Player Game

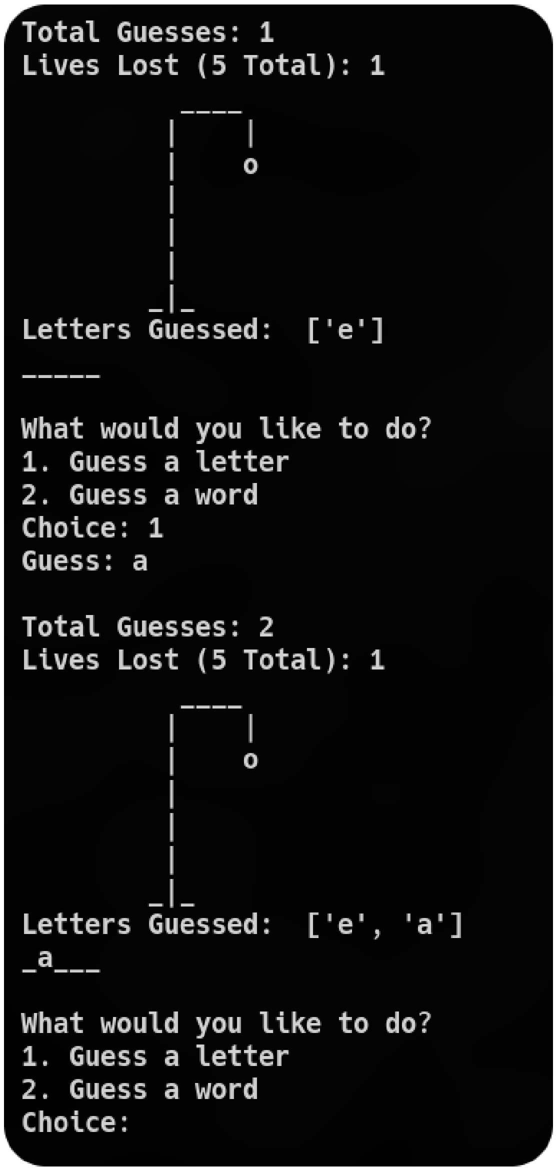
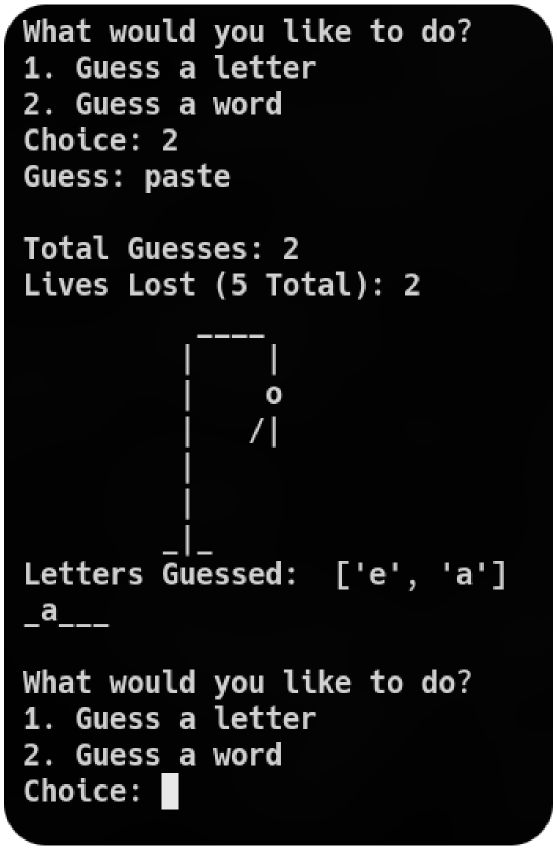
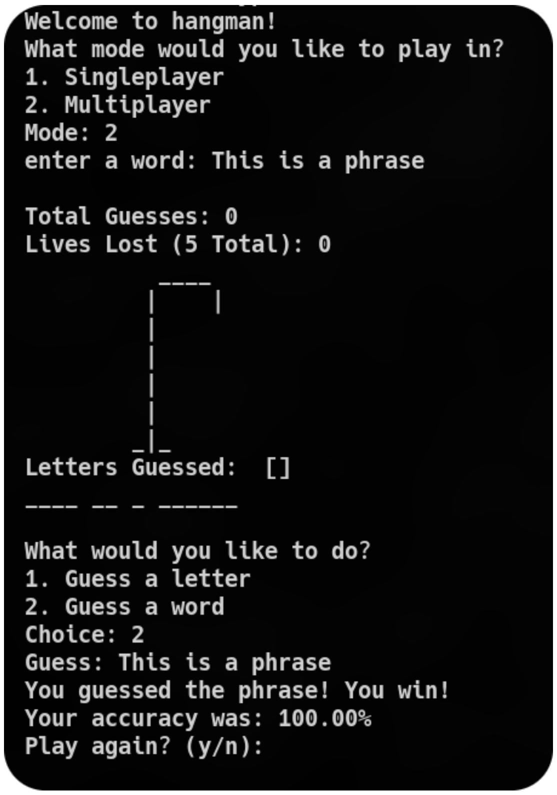

```
Total Guesses: 1
Lives Lost (5 Total): 1
          ____
         |    |
         |    o
         |
         |
         |
       _|_
Letters Guessed:  ['e']
_____

What would you like to do?
1. Guess a letter
2. Guess a word
Choice: 1
Guess: a

Total Guesses: 2
Lives Lost (5 Total): 1
          ____
         |    |
         |    o
         |
         |
         |
       _|_
Letters Guessed:  ['e', 'a']
_a___

What would you like to do?
1. Guess a letter
2. Guess a word
Choice:
```

Figure 2: One Right, One Wrong Guess



```
What would you like to do?
1. Guess a letter
2. Guess a word
Choice: 2
Guess: paste

Total Guesses: 2
Lives Lost (5 Total): 2
          ____
         |    |
         |    o
         |   /|
         |
         |
        _|_
Letters Guessed:  ['e', 'a']
_a___

What would you like to do?
1. Guess a letter
2. Guess a word
Choice: 
```

Figure 3: Guessing a Phrase



```
Welcome to hangman!
What mode would you like to play in?
1. Singleplayer
2. Multiplayer
Mode: 2
enter a word: This is a phrase

Total Guesses: 0
Lives Lost (5 Total): 0

          ____
         |    |
         |
         |
         |
         |
        _|_
Letters Guessed:  []

---- -- - ------

What would you like to do?
1. Guess a letter
2. Guess a word
Choice: 2
Guess: This is a phrase
You guessed the phrase! You win!
Your accuracy was: 100.00%
Play again? (y/n):
```

Figure 4: Multiplayer Game