St Brendan’s Sixth Form College.

CompSci NEA Project

By Peace Olawuyi.

Analysis

Introduction

The project I will be creating is a Python remake of the game “Buckshot Roulette”, also colloquially known as “Buckshot Roulette but Scuffed”.

Description of problem

It will be a free, lightweight rendition of the game “Buckshot Roulette”. It will be a mainly Player vs AI game, where the AI will use algorithms in an attempt to beat the player.

End User Research

* The End Users for this game are:
  + People that don’t really want to buy the game.
  + People who want to test a similar type of game to Buckshot Roulette before committing to buying the official version.
  + People who just like singleplayer games.

6 Measurable Objectives:

1. Live technical stats (mainly for debugging, but also for people who like statistics).
2. High quality menus (such as a main or options / settings menu) and UIs.
3. AI algorithm getting predictions mostly correct.
4. Offline 2 player mode.
5. Customisation (Music / Sounds / Visual Effects).

Design:

Project Design:

A diagram of a game

Description automatically generated

Game Finder

In Game Lobby

Code:

Main.py:

import LogicManager as LM

import Shotgun

#import PlayerUI

import DealerAI as DAI

######################################## GAME ROUNDS ########################################

def GameRounds(GameRound, Lives, ShellCount, ShotgunBalance):

    #Shotgun.ShellCount = ShellCount

    #Shotgun.LoadShotgun(ShellCount, GameRound)

    if GameRound == 1:

        print("\n##### NEW GAME #####")

        print("\n##### ROUND 1 #####")

    if GameRound == 2:

        print("\n\n##### ROUND 2 #####")

    if GameRound == 3:

        print("\n\n##### ROUND 3 #####")

    if GameRound == "End":

        print("\n\n##### GAME OVER #####")

    LM.PlayerLives = Lives # Reset Lives

    LM.DealerLives = Lives

    if ShotgunBalance == False:

        Shotgun.LoadShotgun(ShellCount, False)

    else:

        Shotgun.LoadShotgun(ShellCount, True)

    CurrentShell = LM.CheckCurrentShell()

    while CurrentShell != "Empty":

        CurrentShell = LM.CheckCurrentShell()

        if CurrentShell != "Empty":

            if LM.PlayerLives != 0:

                if LM.DealerLives != 0:

                    LM.PlayersTurn() # Calls for Player's turn.

                else:

                    LM.GUI("PlayerDied", "Dealer") # Report that Dealer died.

                    break

            else:

                LM.GUI("PlayerDied", "Player") # Report that Player died.

                break

        else:

            LM.GUI("Shotgun", "Empty") # Report that shotgun chamber is empty.

            break

        CurrentShell = LM.CheckCurrentShell()

        if CurrentShell != "Empty":

            if LM.DealerLives != 0:

                if LM.PlayerLives != 0:

                    LM.DealersTurn() # Calls for Dealer's turn.

                else:

                    LM.GUI("PlayerDied", "Player") # Report that Player died.

                    break

            else:

                LM.GUI("PlayerDied", "Dealer") # Report that Dealer died.

                break

        else:

            LM.GUI("Shotgun", "Empty") # Report that shotgun chamber is empty.

            break

def StartGame(ShotgunBalance):

    GameRounds(LM.GameRound, 2, 5, ShotgunBalance)

    LM.GameRound += 1

    GameRounds(LM.GameRound, 2, 5, ShotgunBalance)

    #GameRounds(LM.GameRound, 5, 16, ShotgunBalance)

    LM.GameRound += 1

    GameRounds(LM.GameRound, 2, 5, ShotgunBalance)

    #GameRounds(LM.GameRound, 7, 32, ShotgunBalance)

    LM.GameRound = "End"

    GameRounds(LM.GameRound, 0, 0, ShotgunBalance)

######################################## MAIN RUNTIME ########################################

def Main():

    ChooseGameMode = int(input("Choose the Game Mode (1 = Player vs Dealer AI, 2 = Player 1 vs Player 2 (Work In Progress).): ")) # Choose game mode.

    if ChooseGameMode == 2:

        print("\n! Selecting Player 1 vs Player 2 Game Mode. !")

        LM.GameMode = 2

    else:

        print("\n! Selecting Player vs Dealer AI Game Mode. !")

        LM.GameMode = 1

        ChooseAILevel = int(input("\nChoose the Dealer's AI difficulty (1 = Normal, 2 = Hard (Work In Progress).): ")) # Choose the difficulty of the Dealer's AI.

        if ChooseAILevel == 2:

            print("\n! Starting game with 'Hard' Dealer AI diffiiculty. !")

            LM.AILevel = 2

        else:

            print("\n! Starting game with 'Normal' Dealer AI diffiiculty. !")

            LM.AILevel = 1

    ChooseShotgunBalance = int(input("\nDo you want the shotgun to be balanced or not? (1 = No, 2 = Yes.): ")) # Choose whether the shotgun is loading balanced or not.

    if ChooseShotgunBalance == 1:

        print("\n! Starting game with a unbalanced shotgun. !")

        StartGame(False)

    else:

        print("\n! Starting game with an balanced shotgun. !")

        StartGame(True)

Main()

LogicManager.py:

import Shotgun

import PlayerUI

import DealerAI as DAI

######################################## DECLARE VARIABLES ########################################

GameMode = 1

#MaxGameRounds = 3

GameRound = 1

AILevel = 1

CurrentTurn = 0

PlayerLives = 3

DealerLives = 3

#Test = 1

######################################## GUI ########################################

### KEY ###

# ! = System Announcement

# # = Turn Announcement

# - = Action

# () = System "Thoughts"

### KEY ###

def GUI(Element, Modifier):

    if Element == "LiveShell" and Modifier == 0:

        DAI.Track("System", "Live")

        print("! It was a Live. !")

    if Element == "BlankShell" and Modifier == 0:

        DAI.Track("System", "Blank")

        print("! It was a Blank. !")

    if Element == "PlayerDied" and Modifier == "Player":

        print("\n! PLAYER HAS DIED. !")

    if Element == "PlayerDied" and Modifier == "Dealer":

        print("\n! DEALER HAS DIED. !")

    if Element == "Shotgun":

        if Modifier == "Debug": # Basically cheats.

            print("\n", Shotgun.Shotgun)

            print("Current Shell Is:", CheckCurrentShell())

            print("Next Shell Is:", CheckNextShell())

        if Modifier == "Report":

            #GUI("Shotgun", "Debug") # Basically cheats.

            print("\n! There are ", Shotgun.LiveShells, "Live Shells left. !")

            print("! There are ", Shotgun.BlankShells, "Blank Shells left. !")

        if Modifier == "Empty":

            print("\n! Chamber Empty, Skipping Turn. !")

######################################## GAME STUFF ########################################

def CheckCurrentShell():

    if Shotgun.Shotgun[0] == "L":

        return "Live"

    if Shotgun.Shotgun[0] == "B":

        return "Blank"

    if Shotgun.Shotgun[0] == "E" or Shotgun.Shotgun[1] == "":

        return "Empty"

def CheckNextShell():

    if Shotgun.Shotgun[1] == "L":

        return "Live"

    if Shotgun.Shotgun[1] == "B":

        return "Blank"

    if Shotgun.Shotgun[1] == "E" or Shotgun.Shotgun[1] == "":

        return "Empty"

def ShotTaken(Target):

    global PlayerLives

    global DealerLives

    CurrentShell = CheckCurrentShell()

    NextShell = CheckNextShell()

    Shotgun.ShellCount -= 1

    Shotgun.Shotgun.pop(0)

    if Target == "Self" or Target == "Enemy":

        if CurrentShell == "Live":

            GUI("LiveShell", 0)

            Shotgun.LiveShells -= 1

    if Target == "Self":

        if CurrentShell == "Live":

            if CurrentTurn == "Player":

                print("! PLAYER LOST A LIFE. !")

                PlayerLives = PlayerLives - 1

            if CurrentTurn == "Dealer":

                print("! DEALER LOST A LIFE. !")

                DealerLives = DealerLives - 1

    if Target == "Enemy":

        if CurrentShell == "Live":

            if CurrentTurn == "Player":

                print("\n! DEALER LOST A LIFE. !")

                DealerLives = DealerLives - 1

            if CurrentTurn == "Dealer":

                print("\n! PLAYER LOST A LIFE. !")

                PlayerLives = PlayerLives - 1

    if Target == "Enemy":

        if CurrentShell == "Blank":

            GUI("BlankShell", 0)

            Shotgun.BlankShells -= 1

    if Target == "Self":

        if CurrentShell == "Blank":

            GUI("BlankShell", 0)

            Shotgun.BlankShells -= 1

            #print("ShellCount = ", Shotgun.ShellCount)

            if NextShell != "Empty":

                if CurrentTurn == "Player":

                    print("\n! Player gets another go. !")

                    PlayersTurn()

                if CurrentTurn == "Dealer":

                    print("\n! Dealer gets another go. !")

                    DealersTurn()

            #else:

            #    GUI("Shotgun", "Empty")

######################################## PLAYER TURNS ########################################

def PrintLives():

    print("! You have", PlayerLives, "lives remaining. !")

    print("! The Dealer has", DealerLives, "lives remaining. !")

def PlayersTurn():

    global CurrentTurn

    CurrentTurn = "Player"

    GUI("Shotgun", "Report")

    print("\n### Player's Turn:")

    PrintLives()

    Outcome = PlayerUI.Turn()

    #print("Outcome is: ", Outcome)

    if Outcome == "ShootSelf":

        print("\n- You shoot yourself. -")

        ShotTaken("Self")

    elif Outcome == "ShootDealer":

        print("\n- You shoot the Dealer. -")

        ShotTaken("Enemy")

    elif Outcome == "ChoiceFailed":

        PlayersTurn()

    else:

        print("\n! Error. Defaulting to shooting The Dealer. !")

        print("\n- You shoot the Dealer. -")

        ShotTaken("Enemy")

def DealersTurn():

    global CurrentTurn

    CurrentTurn = "Dealer"

    GUI("Shotgun", "Report")

    print("\n### Dealer's Turn:")

    PrintLives()

    Outcome = DAI.Turn(AILevel)

    #print("Outcome is: ", Outcome)

    if Outcome == "ShootSelf":

        print("\n- The Dealer shoots itself. -")

        ShotTaken("Self")

    if Outcome == "ShootPlayer":

        print("\n- The Dealer shoots you. -")

        ShotTaken("Enemy")

DealerAI.py:

import random

import Shotgun

######################################## NOTES ########################################

######## AI Levels ########

# Level 1: AI decides purely on odds. Does not predict future turns.

# Level 2: AI calculates based on odds, and uses player's and its own previous decisions to make guesses on its own next move.

######## HOW THE AI WORKS ########

# Main.py calls for Dealer's turn.

# Turn() receieves Main.py's call.

# If AILevel = 1, Turn() analyses chances and returns its decision.

# If AILevel = 2, Turn() analyses chances and returns its decision to ReturnDecision() (either "Live" or "Blank").

# ReturnDecision() receieves Turn()'s call.

# ReturnDecision() calls Track() to update the list of moves made (see Track() function for details)

# ReturnDecision() returns Turn()'s call to Analyse().

# Analyse() recieves ReturnDecision()'s call.

# Analyse() analyses the past moves kept updated by Track() (see Analyse() function for details).

# Analyse() returns the results of the analysis to ConfirmAnalysis().

# ConfirmAnalysis() recieves Analyse()'s call.

# ConfirmAnalysis() checks the value of Analyse()'s call (the results of the analysis) to ensure that it is within the range laid out by the variables "AnalysisReturnMin" and "AnalysisReturnMax".

# ConfirmAnalysis() compares the value of Analyse()'s call (the results of the analysis) to see whether it is bigger than or equal to the variable "BlankChance" in the event that Turn() returned "Live", or whether it is bigger than or equal to the variable "LiveChance" in the event that Turn() returned "Blank".

# If ConfirmAnalysis() comfirms either of these statements are true, ConfirmAnalysis() will return the decision equal to Turn()'s return. Otherwise, return the opposite decision and update Track().

# ConfirmAnalysis() returns its decision to Main.py.

# Main.py updates LogicManager of Dealer's decision.

# LogicManager updates the game.

######################################## DECLARE VARIABLES ########################################

PlayerDecisions = [] # List of the moves The Player has made.

DealerDecisions = [] # List of the moves The Dealer has made.

CorrectDecision = [] # List of the correct type of shell (generated as each shell is shot.)

AnalysisReturnMax = 100 # Minimum result of analysis for a positive to be returned. (Default = 100).

AnalysisReturnMin = 50 # Minimum result of analysis for a positive to be returned. (Default = 50).

AnalysisResultBuffer = 25

ChancePerShell = 0 # Chance for a shell to be shot.

LiveChance = 0 # Chance for a live shell to be shot.

BlankChance = 0 # Chance for a blank shell to be shot.

######################################## DEBUG STUFF ########################################

def Debug(): # Print variables.

    print("\nChance Per Shell = ", ChancePerShell)

    print("Live Chance:", LiveChance, "%.")

    print("Blank Chance:", BlankChance, "%.")

def AnalysisPrint(): # Prints the lists.

    print("\nPlayer Decisions: ", PlayerDecisions)

    print("Dealer Decisions: ", DealerDecisions)

    print("Correct Decisions: ", CorrectDecision)

######################################## DECISION TRACKER STUFF ########################################

def Track(Target, Decision): # Updates lists as decisions are made and shells are shot.

    if Target == "Player":

        if Decision == "Live":

            PlayerDecisions.append("L")

            DealerDecisions.append("-")

        if Decision == "Blank":

            PlayerDecisions.append("B")

            DealerDecisions.append("-")

    if Target == "Dealer":

        if Decision == "Live":

            DealerDecisions.append("L")

            PlayerDecisions.append("-")

        if Decision == "Blank":

            DealerDecisions.append("B")

            PlayerDecisions.append("-")

    if Target == "System":

        if Decision == "Live":

            CorrectDecision.append("L")

        if Decision == "Blank":

            CorrectDecision.append("B")

######################################## ANALYSIS STUFF ########################################

def ReturnAnalysis(): # Makes a decision depending on results of analysis.

    MoreShotShellType = Analyse() # Analyse past turns to try estimate what shell type has been shot less frequently.

    if MoreShotShellType == "Live": # If blank shells shot less frequently, The Dealer shoots itself.

        print("\n(More lives have been shot than blanks. More likely to be a blank.)")

        print("\n(The Dealer decides to shoot itself.)")

        Track("Dealer", "Blank") # Updates list of previous moves made.

        return "ShootSelf" # Return decision.

    if MoreShotShellType == "Blank": # If live shells shot less frequently, The Dealer shoots The Player.

        print("\n(More blanks have been shot than lives. More likely to be a live.)")

        print("\n(The Dealer decides to shoot you.)")

        Track("Dealer", "Live") # Updates list of previous moves made.

        return "ShootPlayer" # Return decision.

################################################################################

def Analyse(): # Analyses the lists for The Dealer to make a decision. Returns the percentage of confidence.

    if CorrectDecision.count("L") != 0 and DealerDecisions.count("L") != 0 and  PlayerDecisions.count("L") != 0: # Makes sure lists arent empty to avoid divide by zero error.

        DealerCorrectCount = 0

        for i in range (0, len(CorrectDecision)):

            if DealerDecisions[i] == "L" and CorrectDecision[i] == "L":

                DealerCorrectCount = DealerCorrectCount + 1

        PlayerCorrectCount = 0

        for i in range (0, len(CorrectDecision)):

            if PlayerDecisions[i] == "L" and CorrectDecision[i] == "L":

                PlayerCorrectCount = PlayerCorrectCount + 1

        if DealerCorrectCount == 0: # Anti Zero Error. Minimal effect on calculations.

            DealerCorrectCount = 1

        if PlayerCorrectCount == 0:

            PlayerCorrectCount = 1

        #print("\nCorrect Live Count = ", CorrectDecision.count("L"))

        #print("Dealer Correct Count = ", DealerCorrectCount)

        #print("Player Correct Count = ", PlayerCorrectCount)

        ShellCount = CorrectDecision.count("L") # Set "ShellCount" variable.

        LiveAnalysis = ((((DealerCorrectCount + PlayerCorrectCount) / 2) / ShellCount) \* 100)

    else:

        #print("\n(Not enough live shells have been shot to analyse.)")

        LiveAnalysis = 100

    if CorrectDecision.count("B") != 0 and DealerDecisions.count("B") != 0 and  PlayerDecisions.count("B") != 0:

        DealerCorrectCount = 0

        for i in range (0, len(CorrectDecision)):

            if DealerDecisions[i] == "B" and CorrectDecision[i] == "B":

                DealerCorrectCount = DealerCorrectCount + 1

        PlayerCorrectCount = 0

        for i in range (0, len(CorrectDecision)):

            if PlayerDecisions[i] == "B" and CorrectDecision[i] == "B":

                PlayerCorrectCount = PlayerCorrectCount + 1

        if DealerCorrectCount == 0: # Anti Zero Error. Minimal effect on calculations.

            DealerCorrectCount = 1

        if PlayerCorrectCount == 0:

            PlayerCorrectCount = 1

        #print("\nCorrect Blank Count = ", CorrectDecision.count("B"))

        #print("Dealer Correct Count = ", DealerCorrectCount)

        #print("Player Correct Count = ", PlayerCorrectCount)

        ShellCount = CorrectDecision.count("B") # Set "ShellCount" variable.

        BlankAnalysis = ((((DealerCorrectCount + PlayerCorrectCount) / 2) / ShellCount) \* 100)

    else:

        #print("\n(Not enough blank shells have been shot to analyse.)")

        BlankAnalysis = 100

    if LiveAnalysis >= BlankAnalysis:

        return "Live" # More live shells have been shot, current shell more likely to be a blank.

    if BlankAnalysis >= LiveAnalysis:

        return "Blank" # More blank shells have been shot, current shell more likely to be a live.

######################################## DECISION STUFF ########################################

def ReturnDecision(Decision):

    global AnalysisReturnMin

    global AnalysisReturnMax

    if Decision == "Live":

        Analysis = ReturnAnalysis() # Analysing chances of shell being a live.

        #print("Analysis = ", Analysis) # Print the results of the analysis.

        return Analysis # Return decision.

    if Decision == "Blank":

        Analysis = ReturnAnalysis() # Analysing chances of shell being a blank.

        #print("Analysis = ", Analysis) # Print the results of the analysis.

        return Analysis # Return decision.

def Turn(AILevel):

    global ChancePerShell

    global LiveChance

    global BlankChance

    #MaxChoices = 3

    #ChanceShootAI = random.randint(1, MaxChoices)

    #ChanceShootPlayer = random.randint(1, MaxChoices)

    #ChanceUseItem = random.randint(1, MaxChoices)

    ChancePerShell = int(100 / (Shotgun.LiveShells + Shotgun.BlankShells)) # Calculate chance for a shell to be shot.

    LiveChance = ((Shotgun.LiveShells / (Shotgun.LiveShells + Shotgun.BlankShells)) \* 100) # Calculate chance for a live shell to be shot.

    BlankChance = ((Shotgun.BlankShells / (Shotgun.LiveShells + Shotgun.BlankShells)) \* 100) # Calculate chance for a blank shell to be shot.

    #Debug() # Print debug.

    #Outcome = 0

    if AILevel == 1:

        if LiveChance > BlankChance: # If shell more likely to be a live, shoot player.

            return "ShootPlayer" # Return decision.

        if BlankChance > LiveChance: # If shell more likely to be a blank, shoot player.

            return "ShootSelf" # Return decision.

        if LiveChance == BlankChance: # If shell equally likely to be a live or a blank, choose randomly.

            RandomChoice = random.randint(0,1)

            if RandomChoice == 0:

                return "ShootPlayer"

            if RandomChoice == 1:

                return "ShootSelf"

    if AILevel == 2:

        if LiveChance > BlankChance: # If shell more likely to be a live, analyse chance of shell being a live.

            print("\n(The Dealer's initial thought: a Live.)")

            return ReturnDecision("Live") # See function.

        if BlankChance > LiveChance: # If shell more likely to be a blank, analyse chance of shell being a blank.

            print("\n(The Dealer's initial thought: a Blank.)")

            return ReturnDecision("Blank") # See function.

        if LiveChance == BlankChance: # If shell equally likely to be a live or a blank, analyse chances.

            print("\n(After calculating the odds, The Dealer thinks it is an equal chance.)")

            RandomChoice = random.randint(0,1)

            #print("Random Choice = ", RandomChoice)

            if RandomChoice == 0: # "A Live" randomly chosen.

                print("\n(The Dealer's random guess: a Live.)")

                return ReturnDecision("Live") # See function.

            if RandomChoice == 1: # "A Blank" randomly chosen.

                print("\n(The Dealer's random guess: a Blank.)")

                return ReturnDecision("Blank") # See function.

######################################## TESTING STUFF ########################################

def Test():

    global PlayerDecisions

    global DealerDecisions

    global CorrectDecision

    PlayerDecisions = ['E', 'E', 'S', '-', 'E', '-', 'E']

    DealerDecisions = ['-', '-', '-', 'E', '-', 'S', '-']

    CorrectDecision = ['B', 'B', 'L', 'B', 'L', 'L', 'L']

    #Analyse("Live")

    #Analyse("Blank")

#Test()

PlayerUI.py:

import DealerAI as DAI

def Turn():

    Decision = input("\n##### Shoot The Dealer or Yourself? (1 = The Dealer, 2 = You): ")

    if Decision == "1":

        Outcome = "ShootDealer"

        DAI.Track("Player", "Live") # Update list of previous moves made.

        return Outcome

    if Decision == "2":

        Outcome = "ShootSelf"

        DAI.Track("Player", "Blank") # Update list of previous moves made.

        return Outcome

    else:

        return "ChoiceFailed"

Shotgun.py:

import random

# L = Live, B = Blank, E = Empty

ShellTypes = ["L", "B"] # Ensures only Live ("L") or Blank ("B") shells can be loaded into the shotgun.

Shotgun = [] # List of loaded shells.

ShellCount = 0

LiveShells = 0

BlankShells = 0

def SetShotgun():

    global Shotgun

    #Shotgun = ["E", "E", "E", "E", "E", "E", "E", "E", "E", "E", "E", "E", "E", "E", "E", "E", "E"] # 16 Shells + extra "E" as check.

    Shotgun = ["E", "E", "E", "E", "E", "E", "E", "E", "E", "E", "E", "E", "E", "E", "E", "E", "E", "E", "E", "E", "E", "E", "E", "E", "E", "E", "E", "E", "E", "E", "E", "E", "E"] # 32 Shells + extra "E" as check.

def LoadShotgun(ShellNo, Balanced):

    # Shells = How many shells to load

    # GameRound = How to load shotgun depending on the round the game is in.

    # Balanced = Whether loading shells into shotgun is truly random, or balanced (e.g. ensuring that there arent like, 7 lives 1 blank.)

    global ShellTypes

    global Shotgun

    global ShellCount

    global LiveShells

    global BlankShells

    ShellCount = ShellNo

    SetShotgun() # Reset chamber

    if Balanced == False: # Random loading of shotgun.

        for Shell in range(ShellCount):

            Shotgun[Shell] = ShellTypes[random.randint(0, len(ShellTypes) - 1)]

    if Balanced == True: # Balanced loading of shotgun.

        if ShellCount % 2 == 0: # If ShellCount is even.

            Shotgun[0] = "L" # Allows while loop to engage (when shotgun is empty, number of Ls and Bs are technically equal.)

            while Shotgun.count("L") != Shotgun.count("B"): # Loops exits when number of Ls and Bs are equal.

                for Shell in range(ShellCount):

                    Shotgun[Shell] = ShellTypes[random.randint(0, len(ShellTypes) - 1)]

                #print(Shotgun)

        else: # If ShellCount is odd.

            RandomChoice = random.randint(1, 2)

            LiveOrBlank = ""

            if RandomChoice == 1:

                LiveOrBlank = "L" # 1 = Have more Lives, 2 = Have more Blanks

            else:

                LiveOrBlank = "B"

            while Shotgun.count(LiveOrBlank) != ((ShellCount // 2) + 1):

                # Loops exits when number the number of Ls is 1 more than the number of Bs

                #(and vice versa, depending on which letter is chosen).

                for Shell in range(ShellCount):

                    Shotgun[Shell] = ShellTypes[random.randint(0, len(ShellTypes) - 1)]

    LiveShells = Shotgun.count("L")

    BlankShells = Shotgun.count("B")

Testing:

Testing:

Placeholder