**GAMING SYSTEM PROJECT REPORT**

**TABLE OF CONTENTS**

**TOPICS PAGENO.**

**Abstract**

**1. INTRODUCTION 1**

1.1Problem Statement 1

**2. RELATED WORK 2**

**3. ANALYSIS 3**

3.1Software Requirement Specification 3

3.2 System requirements 3

3.2.1Softwarerequirements 4

3.2.2Hardwarerequirements 5

**4. DESIGN 5**

4.1Introduction 5

4.2UMLdiagrams 5

4.2.1 Use case diagram 5

4.2.2Class diagram 6

4.2.3Sequence diagram 6

4.2.4 Activity diagram 7

**5. METHODOLOGY 8**

5.1 Packages used 8

5.2 Concepts used 8

**6. IMPLEMENTATION AND CODE 9**

**7. RESULTS 20**

7.1Output Screens 20

**8. TESTING 22**

8.1Introduction 28

8.2Types of Testing 28

8.3 Test Cases (Min 3) 28

**9. CONCLUSION 24**

**REFERENCES 25**

**ABSTRACT**

The Gaming System is a numerical puzzle game designed to challenge players to reach a target number by adding a number within a customizable range to the previous input. The game offers a unique and engaging gameplay experience suitable for players of all ages and skill levels. It features customizable scales, audio feedback, and a simple user interface, as well as innovative features such as customizable game modes and multiplayer mode, which enhance the overall gaming experience and provide flexibility and variety. The game's customizable scales allow players to choose a range of numbers that can be added to the previous input, giving them control over the level of difficulty. Audio feedback adds a layer of engagement and interactivity to the gameplay experience, notifying the player when they have reached the target number or when they have added an incorrect number. The Gaming System requires a suitable operating system, processor, memory, graphics card, and storage. The game requires the installation of Python and Pygame library and the C graphics. Overall, the Gaming System offers an enjoyable and engaging gameplay experience with customizable features and multiplayer options, making it a great way to challenge oneself and compete with friends and family.

1. **INTRODUCTION**

**1.1 Problem Statement:**

The Gaming System is a software to play the target-scale game where a target T and the scale S are given. The target-scale game involves the use of a customizable range of numbers that can be added to the previous input to reach the target number. The range of numbers can be set by the player, allowing them to choose a level of difficulty that suits their skill level. The game also provides audio feedback, notifying the player when they have reached the target number or when they have added an incorrect number. This feature adds a layer of engagement and interactivity to the gameplay experience.

The Gaming System also includes customizable game modes, which allow the player to choose between different challenges. For example, the game mode can be set to add the same number repeatedly, or it can be set to add a random number within the customizable range. The game also includes a multiplayer mode, which enables players to compete with their friends and family in real-time. This feature adds a social aspect to the game and encourages players to compete against each other.

The Gaming System requires a suitable operating system, processor, memory, graphics card, and storage. The game can be run on Windows, MacOS, or Linux operating systems. The minimum processor requirement is an Intel Core i3 or equivalent. The minimum memory requirement is 4GB of RAM, and the minimum graphics card requirement is an Intel HD Graphics 4000 or equivalent. The game requires at least 100MB of storage space. To run the game, the player will need to install Python and Pygame library and the C graphics. Python is a high-level programming language that is used to develop the game, and Pygame is a set of Python modules used to create games. The C graphics library is used to display graphics in the game.

The Gaming System is a fun and engaging numerical puzzle game that offers a range of customizable features. The game's audio feedback, customizable scales, and game modes provide a unique gameplay experience suitable for players of all ages and skill levels. The game's multiplayer mode also adds a social aspect to the gameplay, making it a great way to compete with friends and family. The system requirements needed to run the game are reasonable and should be accessible to most users. Overall, the Gaming System is a great game that provides hours of entertainment and challenges players to think creatively and strategically.

1. **RELATED WORK**

The concept of numerical games is not new and has been explored by various game developers in the past. Some popular numerical games include 2048, Threes, and Sum Fun, which have gained immense popularity among players of all ages. In this section, we will review some related work in the area of numerical games.

2048 is a popular numerical game that requires players to slide numbered tiles on a grid to combine them and create a tile with the number 2048. The game has gained widespread popularity due to its addictive gameplay, simplicity, and strategic depth.

Threes is another popular numerical game that requires players to slide numbered tiles on a grid to create multiples of three. The game has gained popularity due to its simple gameplay, attractive design, and strategic depth.

Sum Fun is a numerical game that requires players to create equations from a set of tiles. The game has gained popularity due to its unique gameplay experience and educational value, as it helps players develop their mathematical skills.

The Gaming System offers a unique numerical game experience by challenging players to reach a target number by adding a number within a customizable range to the previous input. The game provides a customizable range of numbers and allows players to adjust the range to suit their skill level. Additionally, the game includes audio feedback, customizable game modes, and a multiplayer mode, which enhances the overall gaming experience and adds variety to the gameplay.

In conclusion, numerical games have gained immense popularity among players due to their addictive gameplay and simplicity. The Gaming System offers a unique approach to numerical games by providing a customizable range of numbers, audio feedback, and customizable game modes, which makes it a great addition to the genre.

1. **ANALYSIS**

**3.1 SOFTWARE REQUIRMENTS SPECIFICATION**

The "Gaming System" project is a numerical puzzle game designed to challenge players to reach a target number by adding a number within a customizable range to the previous input. This document outlines the software requirements for the gaming system.

**3.1.1. Functional Requirements:**

* Customizable Scale and Target: The game must allow the player to adjust the range of numbers available for input.
* Audio Feedback: The game must provide audio feedback to the player when an input is correct or incorrect.
* Time Limit: The game must include a time limit for each move, which can be customized by the player.
* Multiplayer Mode: The game must include a multiplayer mode that allows two players to play against each other on the same device.
* Customizable Game Modes: The game must allow players to set their own rules and conditions for playing the game.

**3.1.2. Non-functional Requirements:**

* Usability: The user interface must be simple and intuitive, allowing players to focus on the game's mechanics without any distractions.
* Performance: The game must run smoothly without any lag or delays.
* Compatibility: The game must be compatible with different operating systems and devices.
* Security: The game must not contain any security vulnerabilities or pose a threat to the player's device or data.

**3.1.3. User Requirements:**

* Operating System: The Gaming System should be compatible with a range of operating systems, including Windows, Mac, and Linux, to ensure that players can access the game regardless of their platform.
* Processor: The game should be able to run on a range of processors, including Intel and AMD processors, to ensure that players with different hardware can access the game.
* Storage: The game should have minimum storage requirements to ensure that it does not take up too much space on the player's device.
* Compatibility with Python and Pygame: The game should require the installation of Python and Pygame library to ensure that the game can run correctly.
* Customizable range of numbers: The game should allow players to customize the range of numbers used in the game to suit their skill level.
* Audio feedback: The game should provide audio feedback to the player when they make a move or reach the target number to enhance the gaming experience.

**3.2 SYSTEM REQIREMENTS**

* Operating System: Windows 10 or later, macOS 10.12 or later, or Linux (Ubuntu 16.04 or later)
* Processor: Intel Core i5 or equivalent.
* Memory (RAM): 8GB or higher.
* Graphics Card: NVIDIA GeForce GTX 1050, intel iRISx or equivalent.
* Storage: 10 GB of free space.

**3.2.1 SOFTWARE REQUIRMENTS**

* Python 3.7 or later
* PyGame, sys and random library for python
* Turbo C++ or any editor with installed C graphics.
* IDE or text editor for coding such as PyCharm, Visual Studio Code or any other suitable IDE or editor.

**3.2.2 HARDWARE REQUIRMENTS**

* Processor: Intel Core i5 or equivalent.
* Memory (RAM): 2GB or higher.
* Graphics Card: NVIDIA GeForce GTX 1050 or equivalent. Any graphics card compatible with the operating system and capable of displaying at least 16-bit colour depth. Turbo C++ or any C editor with graphics. h installation.
* Monitor: A display monitor with a minimum resolution of 1024 x 768.
* Input Devices: Keyboard and mouse.
* Audio: A sound card and speakers or headphones.
* Sound Card: Any sound card compatible with the operating system.

1. **DESIGN**

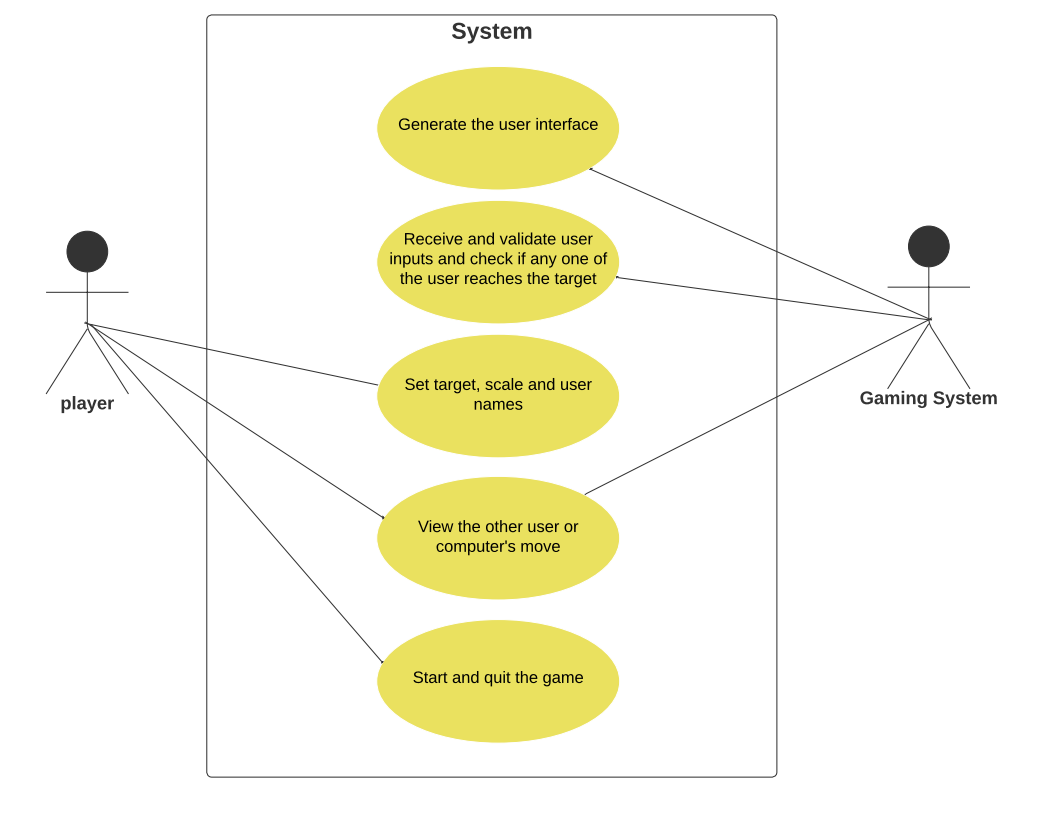
**4.1 INTRODUCTION**

The design of the Gaming System is based on the principles of object-oriented programming, which involves creating objects that interact with each other to provide the desired functionality. The game has been designed using the Unified Modeling Language (UML), which is a standard modeling language used in software engineering to visualize and document software systems.

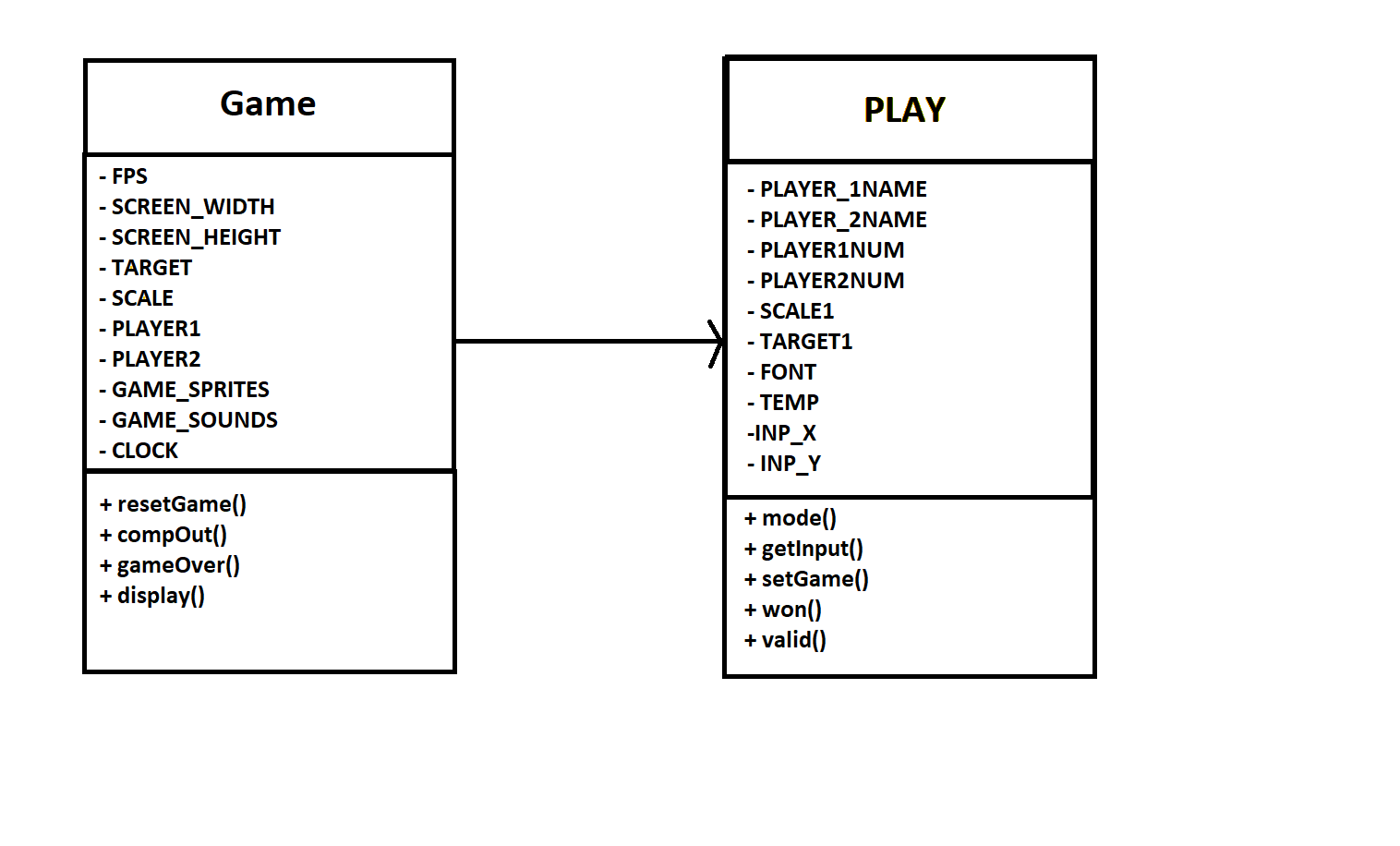
**4.2 UML DIAGRAMS**

The Gaming System has been designed using the Unified Modeling Language (UML), which includes a use case diagram, class diagram, and sequence diagram. These diagrams help to visualize and document the system's functionality, making it easier to understand and maintain.

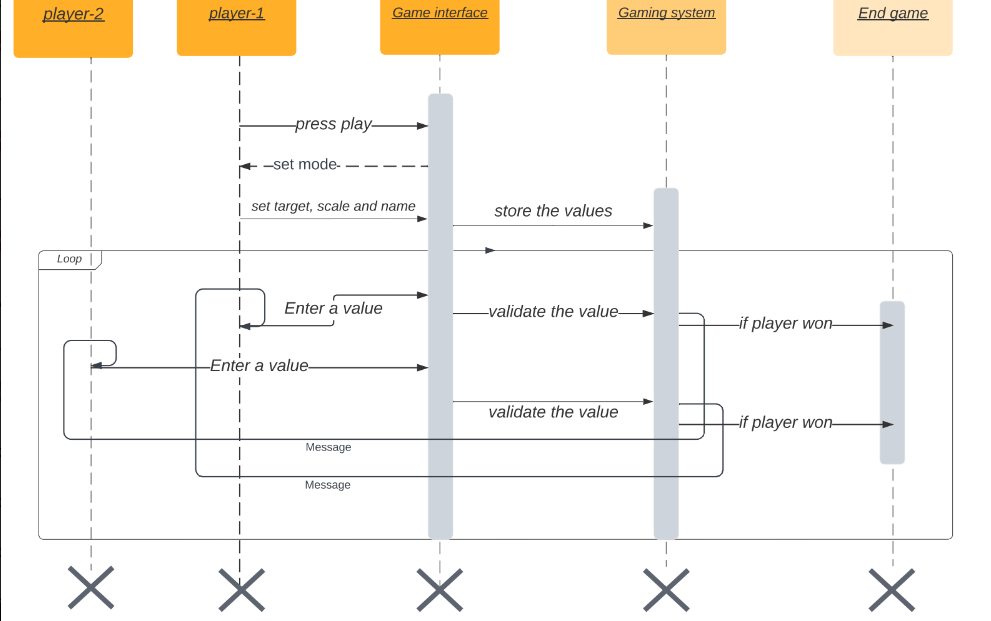
**4.2.1 USE CASE DIAGRAM**

****

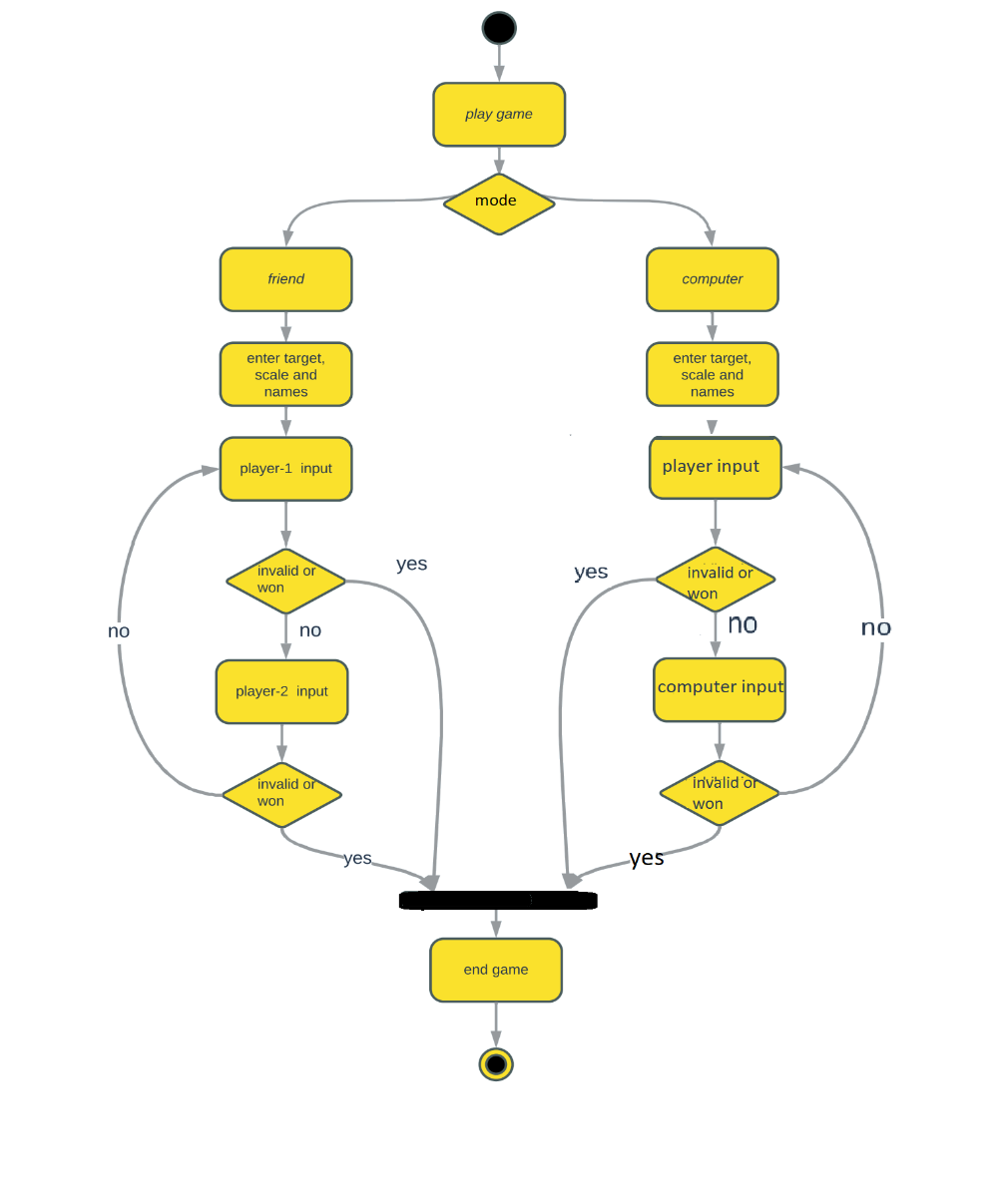
**4.2.2 CLASS DIAGRAM**

****

**4.2.3 SEQUENCE DIAGRAM**

****

**4.2.4 ACTIVITY DIAGRAM**

****

1. **METHODOLOGY**

**5.1 PACKAGES USED**

PyGame is a set of Python modules designed for game development, providing functionality such as graphics, sound, and user input handling. It can be used to create 2D games and is widely used in the game development community.

Sys is a built-in Python module that provides access to system-specific parameters and functions. It can be used in game development to exit the game, handle command-line arguments, and access system-specific resources.

Random is another built-in Python module that provides functions for generating random numbers and sequences. It can be used in game development to generate random events, such as enemy positions, loot drops, or level layouts.

**5.2 CONCEPTS USED**

1. Game mechanics: The rules and systems that govern gameplay, such as scoring, power-ups, and level design.
2. Graphics: The use of visual elements, such as images to create an immersive game environment.
3. Sound design: The use of audio elements, such as music, sound effects to enhance the gameplay experience.
4. User interface (UI): The design of the interface through which players interact with the game, including buttons, and controls.
5. Multiplayer functionality: The ability for multiple players to connect and play together in the same game world.
6. Loops : To repeat the process until an event occurs.
7. Functions: To break down the code into smaller, more manageable pieces. This makes it easier to read, debug and maintain the code, especially as games can involve a lot of complex logic. For Reusability, modularity and abstraction.

**6. IMPLEMENTATION AND CODE**

**Code in C:**

|  |
| --- |
| #include <graphics.h>  #include <conio.h>  #include <stdlib.h>  #include <stdio.h>  #include <time.h>  int mode = 1, target, scale;  char targetstr[5], scalestr[5], p1name[20], p2name[20];  int home()  {  cleardevice();  int ch;  char inp;  // border  setfillstyle(6, 7);  rectangle(0, 0, getmaxx(), getmaxy());  rectangle(0 + 20, 0 + 20, getmaxx() - 20, getmaxy() - 20);  floodfill(1, 1, 15);  settextstyle(0, 0, 5);  outtextxy(65, 60, "Gaming system");  settextstyle(8, 0, 1);  outtextxy(30, 100, "Count Crusade refers to a mission or an effort to reach");  outtextxy(30, 120, " a numerical target by counting or adding numbers. It");  outtextxy(30, 140, "suggests a journey or a battle to achieve a goal through");  outtextxy(60, 160, " the use of numbers and mathematical operations. ");  outtextxy(190, 200, "Press the appropriate key");  setfillstyle(1, 2);  bar3d(198, 260, 398, 300, 5, 1);  floodfill(199, 261, 15);  settextstyle(3, 0, 1);  outtextxy(250, 265, "P - Play");  setfillstyle(1, 9);  bar3d(198, 330, 398, 370, 5, 1);  floodfill(199, 331, 15);  outtextxy(250, 336, "T = Tutorial");  setfillstyle(1, 4);  bar3d(198, 400, 398, 440, 5, 1);  floodfill(199, 401, 15);  outtextxy(250, 403, "Q - Quit");  ch = getch();  return ch;  }  int tutorial()  {  cleardevice();  int ch = 66;  while (1)  {  if (ch == 98 || ch == 66)  {  cleardevice();  setfillstyle(6, 7);  rectangle(0, 0, getmaxx(), getmaxy());  rectangle(0 + 20, 0 + 20, getmaxx() - 20, getmaxy() - 20);  floodfill(1, 1, 15);  setcolor(CYAN);  settextstyle(0, 0, 5);  outtextxy(90, 60, "How to play");  settextstyle(3, 0, 3);  setcolor(15);  outtextxy(40, 125, "Assume:");  outtextxy(40, 145, "Target: 20");  outtextxy(40, 170, "Scale: 1-6");  outtextxy(40, 195, "Players: 2");  outtextxy(40, 215, "Operation: addition(+)");  setcolor(14);  outtextxy(40, 270, "1. Player-1 move must be in a scale of 1 - 6.");  setcolor(GREEN);  rectangle(30, 305, 300, 330);  outtextxy(40, 300, "PLAYER - 1 move : 5");  setcolor(14);  outtextxy(40, 350, "2. Player-2 move must be in a scale of 5 - 11.");  outtextxy(30, 370, "any number from 6(5+1) to 11 (5+6) as scale=6.");  setcolor(GREEN);  rectangle(30, 405, 310, 435);  outtextxy(40, 400, "PLAYER - 2 move : 11");  setcolor(RED);  outtextxy(500, 400, "N");  setcolor(WHITE);  outtextxy(520, 400, "-Next");  setcolor(RED);  outtextxy(500, 425, "H");  setcolor(WHITE);  outtextxy(520, 425, "-Home");  while (1)  {    outtextxy(360, 390, "1 - ");  outtextxy(460, 390, scalestr);  }  // player-1 invalid input  if (flag == 2)  {  outtextxy(90, 350, p1name);  outtextxy(280, 350, "Invalid input !");  if (mode == 2)  outtextxy(140, 390, "Computer");  else  outtextxy(140, 390, p2name);  outtextxy(360, 390, "Won !");  }  // player-2 invalid input  if (flag == 3)  {  outtextxy(90, 350, p2name);  outtextxy(280, 350, "Invalid input !");  outtextxy(140, 390, p1name);  outtextxy(360, 390, "Won !");  }  // player-1 wins  if (flag == 4)  {  outtextxy(140, 350, p1name);  outtextxy(360, 350, "Won !");  }  // player-2 wins  if (flag == 5)  {  if (mode == 2)  outtextxy(140, 350, "Computer");  else  outtextxy(140, 350, p2name);  outtextxy(360, 350, "Won !");  }  if (flag == 2 || flag == 3 || flag == 4 || flag == 5)  {  settextstyle(3, 0, 3);  setcolor(2);  outtextxy(80, 435, "Press any key to exit");  getch();  exit(1);  }  }  int valid(int x, int sum, int id)  {  if (sum - x >= 1 && sum - x <= scale && sum <= target)  return 1;  else  {  if (id == 1)  return 2;  else  return 3;  }  return 1;  }  int won(int x)  {  if (x == target)  return 1;  else  return 0;  }  void play()  {  int player\_1, player\_2;  char p1s[3] = "0", p2s[3] = "0";  display(p1s, p2s, 1);  // player-1 input  scanf("%d", &player\_1);  sprintf(p1s, "%d", player\_1);  display(p1s, p2s, 1);  if (player\_1 < 1 || player\_1 > scale)  {  display(p1s, p2s, 2);  }  while (1)  {  // player-2 input  if (mode == 1)  {  scanf("%d", &player\_2);  sprintf(p2s, "%d", player\_2);  display(p1s, p2s, 1);  }  if (mode == 2)  {  if (player\_1 + scale >= target)  player\_2 = target;  else  {  srand(time(NULL));  player\_2 = rand() % scale + player\_1 + 1;  sprintf(p2s, "%d", player\_2);  display(p1s, p2s, 1);  }  }  if (valid(player\_1, player\_2, 1) == 2)  display(p1s, p2s, 3);  if (won(player\_2))  {  display(p1s, p2s, 5);  }  // player-1 input  scanf("%d", &player\_1);  sprintf(p1s, "%d", player\_1);  display(p1s, p2s, 1);  if (valid(player\_2, player\_1, 2) == 3)  display(p1s, p2s, 2);  if (won(player\_1))  {  display(p1s, p2s, 4);  }  }  }  int rules()  {  cleardevice();  // border  // setfillstyle(6, 7);  rectangle(0, 0, getmaxx(), getmaxy());  rectangle(0 + 20, 0 + 20, getmaxx() - 20, getmaxy() - 20);  // floodfill(5, 5, 15);  setbkcolor(9);  setcolor(2);  settextstyle(0, 0, 4);  outtextxy(230, 50, "RULES");  settextstyle(1, 0, 2);  setcolor(15);  int x = 40, y = 100;  outtextxy(x, y, "1. The objective of the game is to reach a target");  outtextxy(x, y + 30, "number by adding numbers within the scale to the ");  outtextxy(x, y + 60, "previous input.");  outtextxy(x, y + 90, "2. The player starts by inputting a number");  outtextxy(x, y + 120, "within the scale.");  outtextxy(x, y + 150, "3. In each turn, the player must add a new number");  outtextxy(x, y + 180, "within the scale to the previous input.");  outtextxy(x, y + 210, "4. The game ends when the target is reached or when");  outtextxy(x, y + 240, "the player's move is invalid (out of scale).");  outtextxy(x, y + 270, "5. You've press enter after each move.");  setcolor(2);  settextstyle(0, 0, 3);  outtextxy(x + 10, y + 300, "Press any key to start");  getch();  cleardevice();  return 1;  }  int choice()  {  cleardevice();  // border  setfillstyle(1, 12);  rectangle(0, 0, getmaxx(), getmaxy());  rectangle(0 + 20, 0 + 20, getmaxx() - 20, getmaxy() - 20);  floodfill(1, 1, 15);  setfillstyle(10, 4);  floodfill(30, 311, 15);  settextstyle(0, 0, 5);  setcolor(15);  outtextxy(140, 50, "play with");  settextstyle(1, 0, 3);  setcolor(3);  outtextxy(170, 100, "Press the approriate key");  int r;  for (r = 100; r >= 90; r--)  {  setcolor(14);  arc(180, 250, 0, 360, r);  }  settextstyle(8, 0, 3);  outtextxy(130, 230, "F-Friends");  for (r = 100; r >= 90; r--)  {  arc(450, 250, 0, 360, r);  }  settextstyle(8, 0, 3);  outtextxy(380, 230, "C-Computer");  int ch = getch();  if (ch == 70 || ch == 102)  mode = 1;  else if (ch == 99 || ch == 67)  mode = 2;  return 1;  }  int main()  {  int gd = DETECT, gm;  initgraph(&gd, &gm, "C:\\TC\\BGI");  int ch, chp;  ch = home();  while (1)  {  if (ch == 84 || ch == 116)  ch = tutorial();  if (ch == 72 || ch == 104)  ch = home();  if (ch == 80 || ch == 112)  {  choice();  rules();  setgame();  play();  }  else if (ch == 81 || ch == 113)  exit(1);  else  exit(1);  }  getch();  closegraph();  return 0;  } |

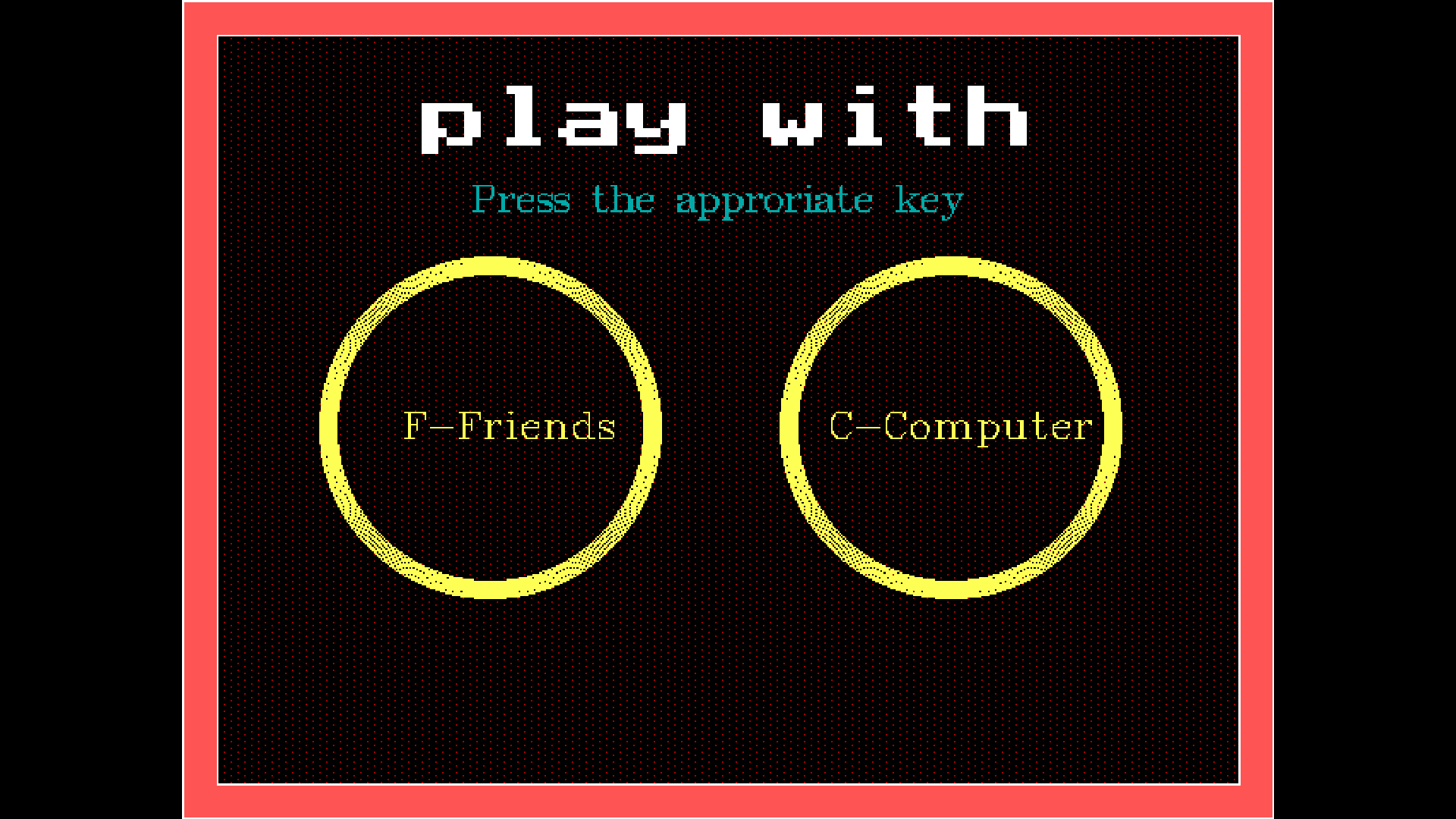
**Code in python:**

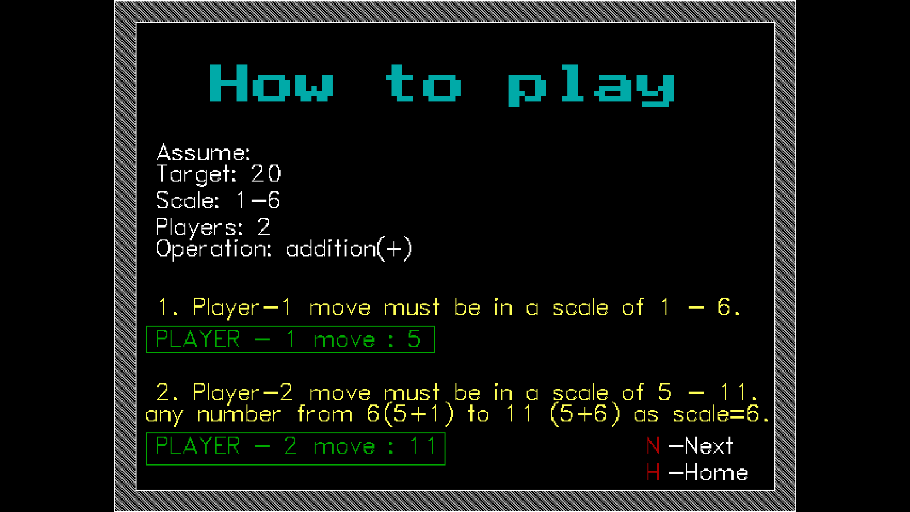
|  |
| --- |
| import pygame  from pygame.locals import \*  import sys  import random  FPS = 32  SCREEN\_WIDTH = 1333  SCREEN\_HEIGHT =750  SCREEN = pygame.display.set\_mode((SCREEN\_WIDTH, SCREEN\_HEIGHT))  TARGET = ''  SCALE = ''  PLAYER\_1Name = ''  PLAYER\_2Name = ''  PLAYER\_1=""  PLAYER\_2=""  PLAYER1NUM = 0  PLAYER2NUM = 0  TARGET1 = 20  SCALE1 = 10  MODE = 1  inp\_x = 600  inp\_y = 420  GAME\_SPRITES = {}  GAME\_SOUNDS = {}  BACKGROUND = pygame.image.load('sprites/background.jpg')  clock = pygame.time.Clock()  clock.tick(30)  def resetGame():  global TARGET, SCALE, PLAYER\_1Name, PLAYER\_2Name, PLAYER\_1, PLAYER\_2, PLAYER1NUM, PLAYER2NUM, TARGET1, SCALE1, inp\_x, inp\_y  TARGET = ''  SCALE = ''  PLAYER\_1Name = ''  PLAYER\_2Name = ''  PLAYER\_1=""  PLAYER\_2=""  PLAYER1NUM = 0  PLAYER2NUM = 0  TARGET1 = 0  SCALE1 = 0  inp\_x = 600  inp\_y = 420  def compOUT(font):  global TARGET  jobdone = False  while not jobdone:  for event in pygame.event.get():  if event.type == pygame.QUIT:  pygame.quit()  quit()  display\_ground(font)  t = str(TARGET1)  player2\_inp = font.render(t, True, (255, 255, 255))  SCREEN.blit(player2\_inp, (920, 500))  pygame.display.update()  jobdone = True  pygame.time.wait(2000)  def gameOver( x):  pygame.time.wait(2000)  font = pygame.font.Font(None, 92)  global PLAYER\_1, PLAYER\_2  SCREEN.fill((255, 255, 255))  p1name = font.render(PLAYER\_1Name, True, (255, 255, 255))  p2name = font.render(PLAYER\_2Name, True, (255, 255, 255))  playAgain\_rect = GAME\_SPRITES['playAgain'].get\_rect()  image\_width, image\_height = GAME\_SPRITES['playAgain'].get\_size()  x1 = (SCREEN.get\_width() - image\_width) - 250  y1 = (SCREEN.get\_height() - image\_height) - 80  while True:  for event in pygame.event.get():  if event.type == pygame.QUIT:  pygame.quit()  quit()  elif event.type == pygame.MOUSEBUTTONDOWN:  mouse\_pos = pygame.mouse.get\_pos()  playAgain\_rect = pygame.Rect(x1, y1, image\_width, image\_height)  if playAgain\_rect.collidepoint(mouse\_pos):  resetGame()  home()  if(x==1):  SCREEN.blit(GAME\_SPRITES['player1Won'], (0, 0))  SCREEN.blit(p1name, (550, 325))  GAME\_SOUNDS['victory'].play()  elif(x==2):  SCREEN.blit(GAME\_SPRITES['player2Won'], (0, 0))  SCREEN.blit(p2name, (550, 325))  GAME\_SOUNDS['victory'].play()  if(x==3):  SCREEN.blit(GAME\_SPRITES['ComputerWon'], (0, 0))  SCREEN.blit(p1name, (510, 325))  SCREEN.blit(GAME\_SPRITES['playAgain'], (x1, y1))  pygame.display.update()  def valid(x, sum):  if sum - x >= 1 and sum - x <= SCALE1 and sum <= TARGET1:  GAME\_SOUNDS['point'].play()  return True  else:  return False  def won(x):  if x == TARGET1:  return True  else:  return False  def get\_user\_input(font, id):  user\_text = ""  inp1\_x = 350  inp1\_y = 500  inp2\_x = 920  inp2\_y = 500  stopwatch\_duration = 7000  stopwatch\_start\_time = pygame.time.get\_ticks()  while True:  for event in pygame.event.get():  if event.type == pygame.QUIT:  pygame.quit()  quit()  # if MODE==1:  if id==1:  SCREEN.blit(text\_surface, (inp1\_x, inp1\_y))  else:  SCREEN.blit(text\_surface, (inp2\_x, inp2\_y))  pygame.display.update()  if elapsed\_time >= stopwatch\_duration:  if(MODE==1):  gameOver(3)  gameOver(id)  def display\_ground(font):  global PLAYER\_1, PLAYER\_2  inp1\_x = 350  inp1\_y = 500  inp2\_x = 920  inp2\_y = 500  p1name = font.render(PLAYER\_1Name, True, (255, 255, 255))  p2name = font.render(PLAYER\_2Name, True, (255, 255, 255))  if MODE==1:  SCREEN.blit(GAME\_SPRITES['mode1\_screen'], (0, 0))  SCREEN.blit(p1name, (270, 350))  SCREEN.blit(p2name, (870, 350))  else:  SCREEN.blit(GAME\_SPRITES['mode2\_screen'], (0, 0))  SCREEN.blit(p1name, (270, 350))  SCREEN.blit(p2name, (870, 350))  def play():  global TARGET, SCALE, PLAYER\_1Name, PLAYER\_2Name, PLAYER\_1, PLAYER\_2, PLAYER1NUM, PLAYER2NUM, TARGET1, SCALE1  font = pygame.font.Font(None, 72)  global PLAYER\_1, PLAYER\_2  display\_ground(font)  PLAYER\_1 = get\_user\_input(font, 1)  PLAYER1NUM = int(PLAYER\_1)  display\_ground(font)  if PLAYER1NUM < 1 or PLAYER1NUM > SCALE1:  if MODE != 1:  gameOver(2)  else:  gameOver(3)  while True:  display\_ground(font)  player1\_inp = font.render(PLAYER\_1, True, (255, 255, 255))  SCREEN.blit(player1\_inp, (350, 500))  SCREEN.blit(player2\_inp, (920, 500))  PLAYER\_1 = get\_user\_input(font, 1)  PLAYER1NUM = int(PLAYER\_1)  if not valid(PLAYER2NUM, PLAYER1NUM):  if MODE != 1:  gameOver(2)  else:  gameOver(3)  if valid(PLAYER2NUM, PLAYER1NUM) and won(PLAYER1NUM):  gameOver( 1)  def home():  SCREEN.fill((255, 255, 255))  image\_rect = GAME\_SPRITES['playBtn'].get\_rect()  image\_width, image\_height = GAME\_SPRITES['playBtn'].get\_size()  x2 = (SCREEN.get\_width() - image\_width) / 2  y2 = (SCREEN.get\_height() - image\_height) / 2 + 150  while True:  for event in pygame.event.get():  if event.type == QUIT or (event.type == KEYDOWN and event.key == K\_ESCAPE):  pygame.quit()  sys.exit()  elif event.type == pygame.MOUSEBUTTONDOWN:  mouse\_pos = pygame.mouse.get\_pos()  image\_rect = pygame.Rect(x2, y2, image\_width, image\_height)  if image\_rect.collidepoint(mouse\_pos):  mode()  GAME\_SOUNDS['start'].play()  SCREEN.blit(GAME\_SPRITES['homeImg'], (0, 0))  SCREEN.blit(GAME\_SPRITES['playBtn'], (x2, y2))  pygame.display.update()  def mode():  SCREEN.fill((255, 255, 255))  image\_rect1 = GAME\_SPRITES['compBtn'].get\_rect()  image\_rect2 = GAME\_SPRITES['friendsBtn'].get\_rect()  if comp\_rect.collidepoint(mouse\_pos):  MODE = 1  GAME\_SOUNDS['start'].stop()  setGame()  pygame.display.update()  def getInput(font, id):  user\_text = ""  global inp\_y  size = (400, 300)  line\_color = (0, 0, 0)  line\_x = size[0] // 2  line\_width = 3  line\_height = 50  blink\_rate = 20  blink\_timer = blink\_rate // 2  clicked = False  nextBtn\_rect = GAME\_SPRITES['nextBtn'].get\_rect()  image\_width, image\_height = GAME\_SPRITES['nextBtn'].get\_size()  x1 = (SCREEN.get\_width() - image\_width) - 100  y1 = (SCREEN.get\_height() - image\_height) - 80  backBtn\_rect = GAME\_SPRITES['backBtn'].get\_rect()  image\_width1, image\_height1 = GAME\_SPRITES['backBtn'].get\_size()  x2 = 80  y2 = (SCREEN.get\_height() - image\_height1) - 80  if not clicked:  blink\_timer -= 1  if blink\_timer == 0:  blink\_timer = blink\_rate  if line\_color == (0, 0, 0):  line\_color = (255, 255, 255)  else:  line\_color = (0, 0, 0)  pygame.display.update()  def setGame():  global inp\_x, id, PLAYER\_1Name, PLAYER\_2Name, TARGET, TARGET1, SCALE, SCALE1  font = pygame.font.Font(None, 72)  TARGET = getInput( font, 1)  TARGET1 = int(TARGET)  SCALE = getInput( font, 2)  SCALE1 = int(SCALE)  inp\_x -= 100  PLAYER\_1Name = getInput( font, 3)  if(MODE == 2):  PLAYER\_2Name = getInput( font, 4)  play()  else:  PLAYER\_2Name = "Computer"  play()  if \_name\_ == "\_main\_":  pygame.init()  FPSCLOCK = pygame.time.Clock()  pygame.display.set\_caption('GAMING SYSTEM')  base\_font = pygame.font.SysFont('Calibri', 25)  GAME\_SPRITES['playBtn'] =pygame.image.load('sprites/playButton.png').convert\_alpha()  GAME\_SPRITES['nextBtn'] =pygame.image.load('sprites/nextBtn.png').convert\_alpha()  GAME\_SPRITES['backBtn'] =pygame.image.load('sprites/backBtn.png').convert\_alpha()  GAME\_SPRITES['compBtn'] =pygame.image.load('sprites/compBtn.png').convert\_alpha()  GAME\_SPRITES['friendsBtn'] =pygame.image.load('sprites/friendsBtn.png').convert\_alpha()  GAME\_SPRITES['mode1'] =pygame.image.load('sprites/mode1.png').convert\_alpha()    GAME\_SPRITES['player2Won'] = pygame.image.load('sprites/player2Won.png').convert\_alpha()  GAME\_SPRITES['ComputerWon'] = pygame.image.load('sprites/ComputerWon.png').convert\_alpha()  GAME\_SPRITES['getScale'] = pygame.image.load('sprites/scale.png')  GAME\_SPRITES['getP1Name'] = pygame.image.load('sprites/p1Name.png')  GAME\_SPRITES['getP2Name'] = pygame.image.load('sprites/p2Name.png')  GAME\_SPRITES['playAgain'] = pygame.image.load('sprites/playAgain.png')  GAME\_SPRITES['numbers'] = (  pygame.image.load('sprites/0.png').convert\_alpha(),  pygame.image.load('sprites/1.png').convert\_alpha(),  pygame.image.load('sprites/2.png').convert\_alpha(),  pygame.image.load('sprites/3.png').convert\_alpha(),  pygame.image.load('sprites/4.png').convert\_alpha(),  pygame.image.load('sprites/5.png').convert\_alpha(),  pygame.image.load('sprites/6.png').convert\_alpha()  )    GAME\_SOUNDS['start'] = pygame.mixer.Sound('audio/start.mp3')  GAME\_SOUNDS['victory'] = pygame.mixer.Sound('audio/victory.mp3')  GAME\_SOUNDS['start1'] = pygame.mixer.Sound('audio/start1.mp3')  GAME\_SOUNDS['point'] = pygame.mixer.Sound('audio/point.wav')  home() |

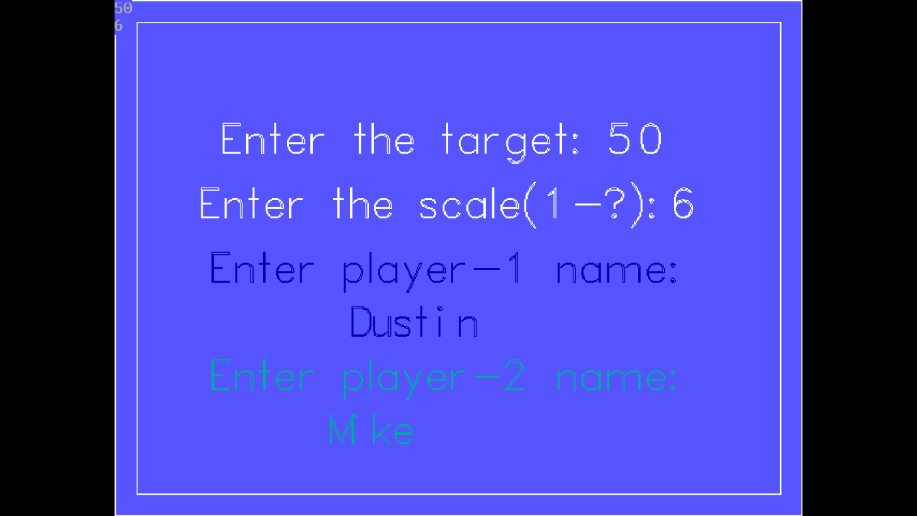
**7. RESULTS**

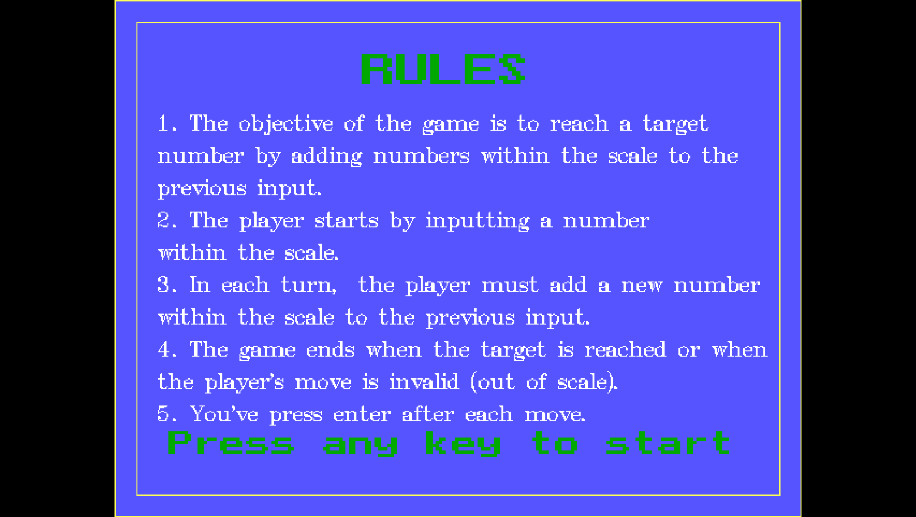
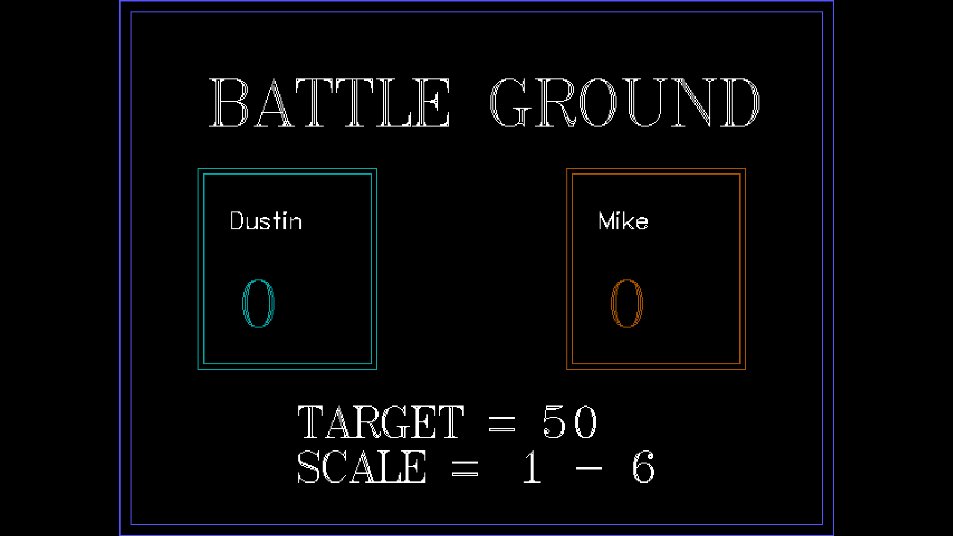
**7.1 OUTPUT SCREENS**

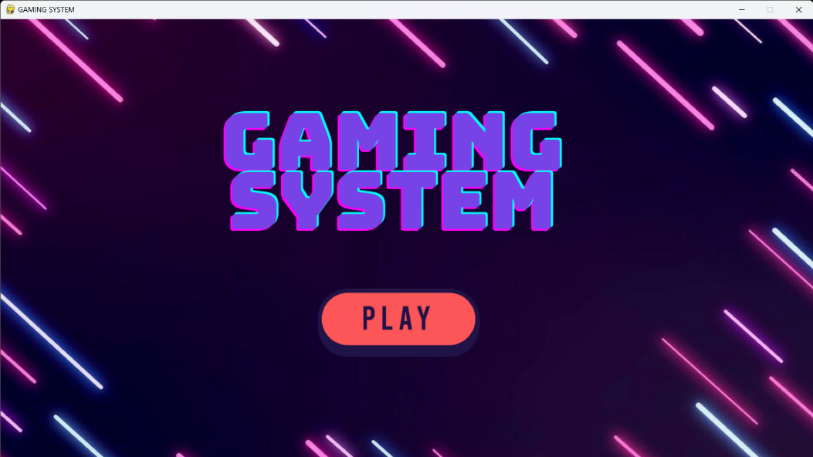
**Using C Language:**

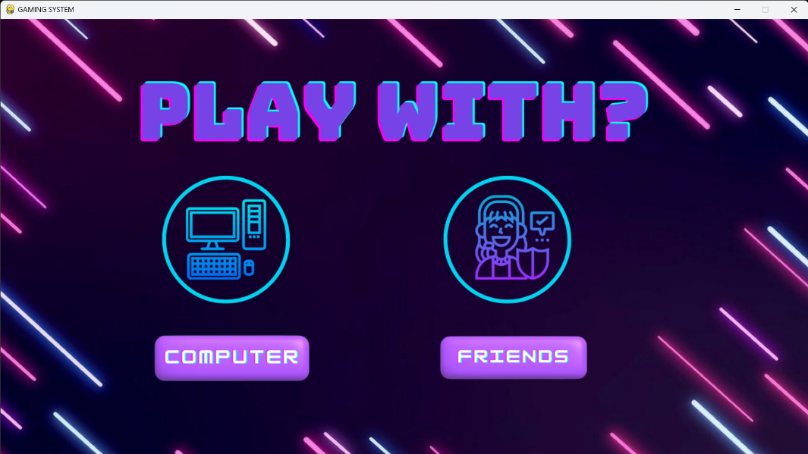
****

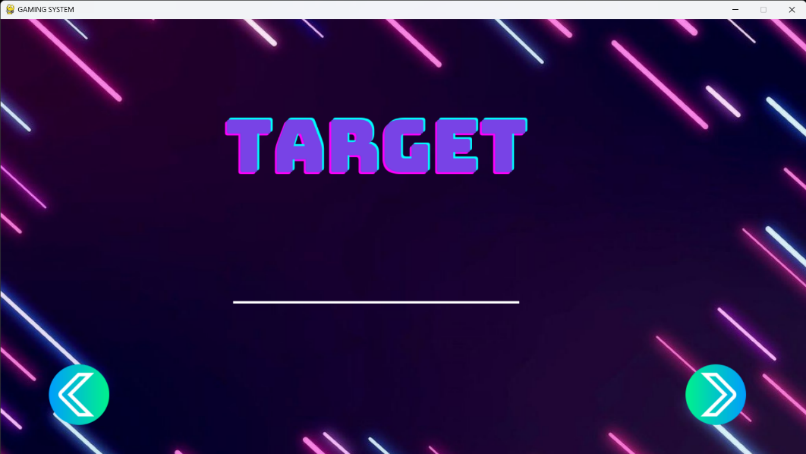
****

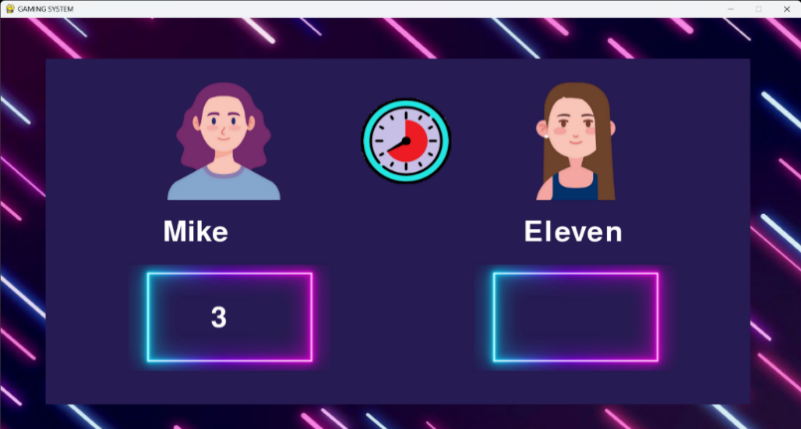
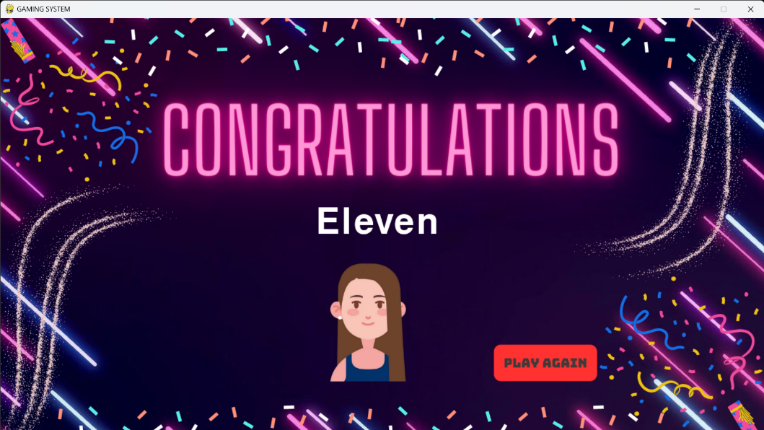
****

****

**Using Python:**

****

****

****

**8.TESTING**

**8.1. Types of Testing**

There are several types of testing that were performed on the "Gaming System" project to ensure its functionality and quality:

1. Unit Testing: Testing individual components of the game, such as the input system, score calculation, and audio feedback, to ensure they work correctly.
2. Integration Testing: Testing how different components of the game work together, such as how the input system interacts with the score calculation.
3. Functional Testing: Testing the overall functionality of the game to ensure it meets the specified requirements, such as reaching the numerical target by adding numbers within the scale.
4. Performance Testing: Testing the game's performance under different conditions, such as a high number of concurrent players or low network bandwidth.
5. Usability Testing: Testing the user interface and user experience of the game to ensure it is intuitive and easy to use.
6. Compatibility Testing: Testing the game on different hardware and software configurations to ensure it works on a variety of platforms and devices.
7. Security Testing: Testing the game for potential security vulnerabilities, such as data breaches or hacking attempts.

**8.2 Test cases:**

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Test case id | Test title | Test data | Testing steps | Expected output | Actual output | status |
| 1. | Start Game | Click on the "Start Game" button | Click on the "Start Game" button | The mode screen should appear. | Navigated to mode screen. | pass |
| 2. | check for background music and sound effects | On/off sound &background music | 1.Verify if sound effects are in sync with action  2.on/off device sound (native sound) and check for vibration effect if present | Sound effects are sync with action and vibration effect is present | Background music and sound effects are successfully executed | pass |
| 3 | Performance | The Loading time of a game | to check that any action is taking considerable time or not and game flow should be fast or not | Action taking fast and game flow is also fast | Game flow taking fast | pass |
| 4. | Scale not entered | No scale is given | To check game when scale isn’t given | Game should stop with a prompt | Scale is taken 0 | fail |
| 5. | Score | Score calculation | To check winner of the game person or computer | Person  Get the score | Person get the score | pass |
| 6. | Time out | Doing actions within the time | To check given actions are doing within the time or not | Actions done within the time | Actions done within the time | pass |
| 7. | multitasking | Switch b/w different apps and play game | To check switching option is available or not | Switch b/w different apps and play game | Switch option is successfully executed | pass |
| 8. | functionality | Menu options and different game modes/location | To check for the features that will be unlocked level-wise | Unlocked level-wise | Locked level-wise | fail |
| 9. | Device and os | To support screen sizes and os versions | To check os is supported or not | Os supported | Os(versions) are supported | pass |

**9. Conclusion**

In conclusion, the "Gaming System" project is a target scale game that challenges players to reach a numerical target by adding a number within a customizable range to the previous input. The game offers several unique and innovative features, such as customizable scales, time limits, and multiplayer modes, providing players with a dynamic and engaging gameplay experience.

Throughout the development process, the project has encountered several challenges, such as developing a reliable input system and implementing high-quality audio and graphics. However, these obstacles were overcome through careful planning, testing, and iteration, resulting in a high-quality game that meets the specified requirements.

To ensure the game's functionality, performance, usability, and security, a thorough testing process was conducted, including various types of testing, such as unit testing, integration testing, functional testing, performance testing, usability testing, compatibility testing, and security testing.

Overall, the "Gaming System" project offers an exciting and challenging game that is enjoyable for players of all ages and skill levels. Its unique features, customizable settings, and engaging gameplay make it an excellent addition to the gaming industry, providing players with a new and exciting way to enjoy their free time.

**References**

<https://www.pygame.org>

<https://pypi.org/project/pygame>

"Making Games with Python & Pygame" by Al Sweigart

Beginning Game Programming with Pygame Zero by Stewart Watkiss

“ Computer Graphics Using OpenGL” by F.S. Hill Jr.