1. CƠ SỞ LÝ THUYẾT
   1. GAME NỐI THÚ POKEMON
      1. Lịch sử

Trò chơi "Nối thú Pokémon" là một trò chơi điện tử giải đố được phát hành bởi hãng game Nhật Bản là Genius Sonority vào ngày 18/02/2015. Trò chơi này được xây dựng trên thế giới nhân vật của hoạt hình Pokémon nổi tiếng. Genius Sonority là một công ty phát triển trò chơi có trụ sở tại Tokyo, Nhật Bản. Họ đã tạo ra nhiều trò chơi dựa trên thương hiệu Pokémon và đã góp phần làm phong phú thêm hệ thống game của thế giới Pokémon. Trò chơi "Nối thú Pokémon" là một trong những sản phẩm của hãng, mang đến trải nghiệm giải đố thú vị và hấp dẫn cho người chơi.

* + 1. Mô tả

Trò chơi "Nối thú Pokémon" là một trò chơi nối 2 hình ảnh Pokémon giống nhau và xóa chúng đi, với điều kiện đường nối 2 hình ảnh Pokémon có số lần gấp khúc không quá 2. Đường nối không được vượt qua bất kỳ hình ảnh hoặc vật cản nào khác và chỉ được di chuyển theo hàng ngang hoặc hàng dọc, không được đi chéo.

Trò chơi cung cấp nhiều cách sắp xếp Pokémon khác nhau, bao gồm đứng yên, lên trên, xuống dưới, sang trái, sang phải, về giữa và theo đường chéo. Mỗi cấp độ tương ứng với một cách sắp xếp Pokémon.

Sau mỗi lần xóa Pokémon thành công, nếu không còn cặp Pokémon nào có thể xóa, các hình ảnh Pokémon sẽ được sắp xếp lại tự động (số lần sắp xếp lại vị trí giới hạn là 9). Trò chơi cũng cung cấp gợi ý về cặp Pokémon có thể xóa (số lần gợi ý giới hạn là 2).

Trò chơi kết thúc trong các trường hợp sau: hết thời gian quy định, không còn khả năng sắp xếp lại vị trí hoặc đã chơi hết tất cả các cấp độ. Người chơi có thể lựa chọn chơi lại cấp độ đã hoàn thành sau mỗi cấp độ.

Điểm số được tính dựa trên số cặp Pokémon đã xóa và thời gian còn lại sau mỗi cấp độ. Công thức tính điểm là: Điểm=(số cặp đã xóa)\*10+(thời gian còn lại ở mỗi cấp độ)\*2.

Giao diện trò chơi bao gồm một lưới các hình ảnh Pokémon, hiển thị điểm số, nút sắp xếp lại vị trí, nút gợi ý và hiển thị thời gian.

1. KẾT QUẢ NGHIÊN CỨU
   1. TÌM ĐƯỜNG ĐI GIỮA 2 POKEMON BẰNG DEQUE VÀ BFS
      1. Ý tưởng

Bước 1: Gán giá trị ban đầu cho f[x\_start][y\_start][dir] là 1 và thêm tọa độ và hướng di chuyển của Cell đầu vào hàng đợi.

Bước 2: Trong quá trình duyệt hàng đợi:

Lấy phần tử đầu tiên trong hàng đợi (t).

Kiểm tra điều kiện dừng: nếu tọa độ của t trùng khớp với tọa độ điểm b và f[t[0]][t[1]][t[2]] thuộc đoạn từ 1 đến 3, thoát khỏi vòng lặp.

Duyệt qua 4 hướng di chuyển:

Tính toán tọa độ mới (x, y) dựa trên tọa độ hiện tại và hướng di chuyển.

Kiểm tra xem tọa độ mới có nằm trong giới hạn của lưới và có thể di chuyển vào ô đó không.

Nếu điều kiện được thỏa mãn, kiểm tra hướng di chuyển:

Nếu hướng di chuyển khác với hướng di chuyển của ô hiện tại và f[x][y][dir] > f[t[0]][t[1]][t[2]] + 1, cập nhật f[x][y][dir] và thêm tọa độ mới và hướng di chuyển vào cuối hàng đợi.

Nếu hướng di chuyển trùng khớp với hướng di chuyển của ô hiện tại và f[x][y][dir] > f[t[0]][t[1]][t[2]], cập nhật f[x][y][dir] và thêm tọa độ mới và hướng di chuyển vào đầu hàng đợi.

Cập nhật mảng trace với tọa độ nguồn của tọa độ mới.

Bước 3: Sau khi kết thúc vòng lặp, duyệt qua 4 hướng di chuyển và kiểm tra xem có đường đi từ Cell sau đến các điểm khác hay không, nếu có thì thực hiện cập nhật đường đi và hình vẽ nối đường đi (flag=True).

* + 1. Cài đặt

def has\_path(self, a:Cell, b:Cell, flag):

      f = [[[self.INT\_MAX] \* 4 for \_ in range(self.N\_COL + 2)] for \_ in range(self.N\_ROW + 2)]

      trace = [[[(-1, -1, -1)] \* 4 for \_ in range(self.N\_COL + 2)] for \_ in range(self.N\_ROW + 2)]

      queue = []

      d = [(1, 0), (0, 1), (-1, 0), (0, -1)]

      x\_start, y\_start, x\_end, y\_end = a.r, a.c, b.r, b.c

      for dir in range(4):

        f[x\_start][y\_start][dir] = 1

        queue.append((x\_start, y\_start, dir))

      while len(queue) > 0:

        t = queue[0]

        queue.pop(0)

        if (t[0], t[1]) == (x\_end, y\_end) and 1 <= f[t[0]][t[1]][t[2]] <= 3: break

        for dir, t1 in enumerate(d):

          x = t[0] + t1[0]

          y = t[1] + t1[1]

          if self.inside(x, y) and (self.cell[x][y].val\_pokemon == 0 or (x, y) == (x\_end, y\_end)):

            if dir != t[2] and f[x][y][dir] > f[t[0]][t[1]][t[2]] + 1:

              f[x][y][dir] = f[t[0]][t[1]][t[2]] + 1

              queue.append((x, y, dir))

              trace[x][y][dir] = (t[0], t[1], t[2])

            if dir == t[2] and f[x][y][dir] > f[t[0]][t[1]][t[2]]:

              f[x][y][dir] = f[t[0]][t[1]][t[2]]

              queue.insert(0, (x, y, dir))

              trace[x][y][dir] = (t[0], t[1], t[2])

      for dir in range(4):

        if 1 <= f[x\_end][y\_end][dir] <= 3:

          if flag == True:

            x, y, d = x\_end, y\_end, dir

            while True:

              t0 = (x, y, d)

              t1 = trace[x][y][d]

              t2 = trace[t1[0]][t1[1]][t1[2]]

              if (t1[0], t1[1]) == (x\_start, y\_start):

                return True

              v1 = (t1[0]-t0[0], t1[1]-t0[1])

              v2 = (t2[0]-t1[0], t2[1]-t1[1])

              val = self.get\_path\_value(v1, v2)

              self.cell[t1[0]][t1[1]].update\_path\_val(val, self.img\_line\_path[val])

              self.delete\_queue.append((time.time(), self.cell[t1[0]][t1[1]]))

              x, y, d = t1[0], t1[1], t1[2]

          return True

      return False

* 1. XÓA CẶP POKEMON

def delete\_cell(self):

while len(self.delete\_queue) > 0:

        t = self.delete\_queue[0]

        if(time.time() - t[0] > self.delete\_time\_wait):

          if len(t) == 3:

            self.change\_state(self.level, t[1], t[2])

          else:

            t[1].delete()

          self.delete\_queue.pop(0)

        else:

          Return

 def change\_state(self, level:int, c1:Cell, c2:Cell):

      if level == 0:

        c1.delete()

        c2.delete()

      elif level == 1: #rot xuong

        self.fall\_down1(c1, c2)

      elif level == 2: #rot len

        self.fall\_up1(c1, c2)

      elif level == 3: #rot trai

        self.fall\_left1(c1, c2)

      elif level == 4: #rot phai

        self.fall\_right1(c1, c2)

      elif level == 5: #rot vo giua theo chieu ngang

        if 2\*c1.r <= self.N\_ROW and 2\*c2.r <= self.N\_ROW:

          self.fall\_down1(c1, c2)

        elif 2\*c1.r > self.N\_ROW and 2\*c2.r > self.N\_ROW:

          self.fall\_up1(c1, c2)

        else:

          if c1.r < c2.r:

            self.fall\_down(c1)

            self.fall\_up(c2)

          else:

            self.fall\_up(c1)

            self.fall\_down(c2)

      elif level == 6: #rot vo giua theo chieu doc

        if 2\*c1.c <= self.N\_COL and 2\*c2.c <= self.N\_COL:

          self.fall\_right1(c1, c2)

        elif 2\*c1.c > self.N\_COL and 2\*c2.c > self.N\_COL:

          self.fall\_left1(c1, c2)

        else:

          if c1.c < c2.c:

            self.fall\_right(c1)

            self.fall\_left(c2)

          else:

            self.fall\_left(c1)

            self.fall\_right(c2)

      elif level == 7: #rot sang 2 ben theo chieu doc

        if 2\*c1.c <= self.N\_COL and 2\*c2.c <= self.N\_COL:

          self.fall\_left1(c1, c2, False)

        elif 2\*c1.c > self.N\_COL and 2\*c2.c > self.N\_COL:

          self.fall\_right1(c1, c2, False)

        else:

          if c1.c < c2.c:

            self.fall\_left(c1, False)

            self.fall\_right(c2, False)

          else:

            self.fall\_right(c1, False)

            self.fall\_left(c2, False)

      elif level == 8: #rot 2 ben theo chieu ngang

        if 2\*c1.r <= self.N\_ROW and 2\*c2.r <= self.N\_ROW:

          self.fall\_up1(c1, c2, False)

        elif 2\*c1.r > self.N\_ROW and 2\*c2.r > self.N\_ROW:

          self.fall\_down1(c1, c2, False)

        else:

          if c1.r < c2.r:

            self.fall\_up(c1, False)

            self.fall\_down(c2, False)

          else:

            self.fall\_down(c1, False)

            self.fall\_up(c2, False)

      elif level == 9:

        self.fall\_left\_up1(c1, c2)

      elif level == 10:

        self.fall\_left\_down1(c1, c2)

      elif level == 11:

        self.fall\_right\_up1(c1, c2)

      elif level == 12:

        self.fall\_right\_down1(c1, c2)

      if c2.val\_pokemon:

        c2.set\_chosen(False)

        c2.set\_can\_click(True)

        c2.unclick()

      if c1.val\_pokemon:

        c1.set\_chosen(False)

        c1.set\_can\_click(True)

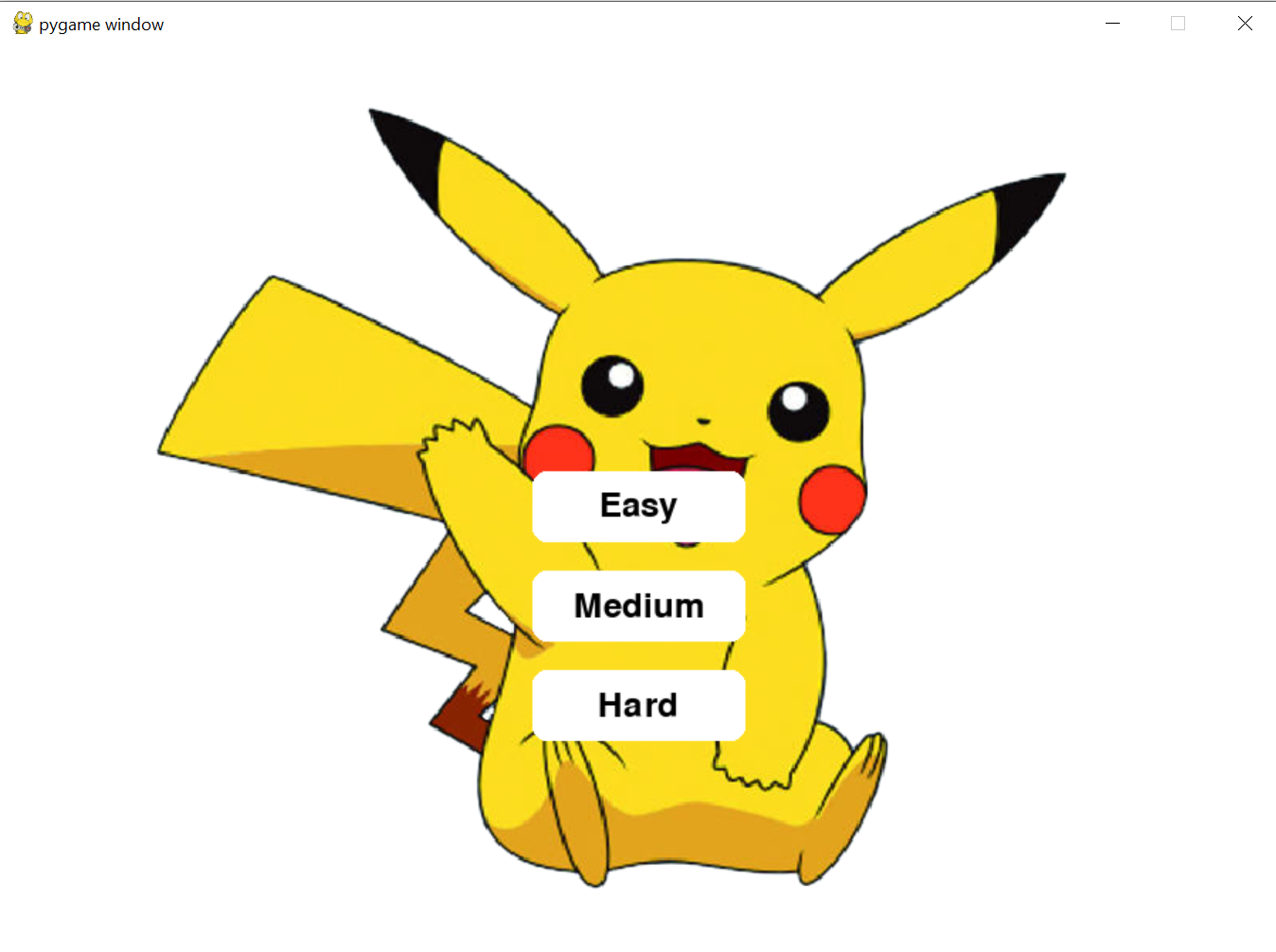
        c1.unclick()

      self.current\_cell -= 2

      if self.is\_sound:

        self.sound\_conected.play()

* 1. GIAO DIỆN
     1. Giao diện trang chủ



Giao diện trang chủ

* + 1. Giao diện màn hình chơi

A screenshot of a video game

Description automatically generated

Giao diện màn hình chơi game

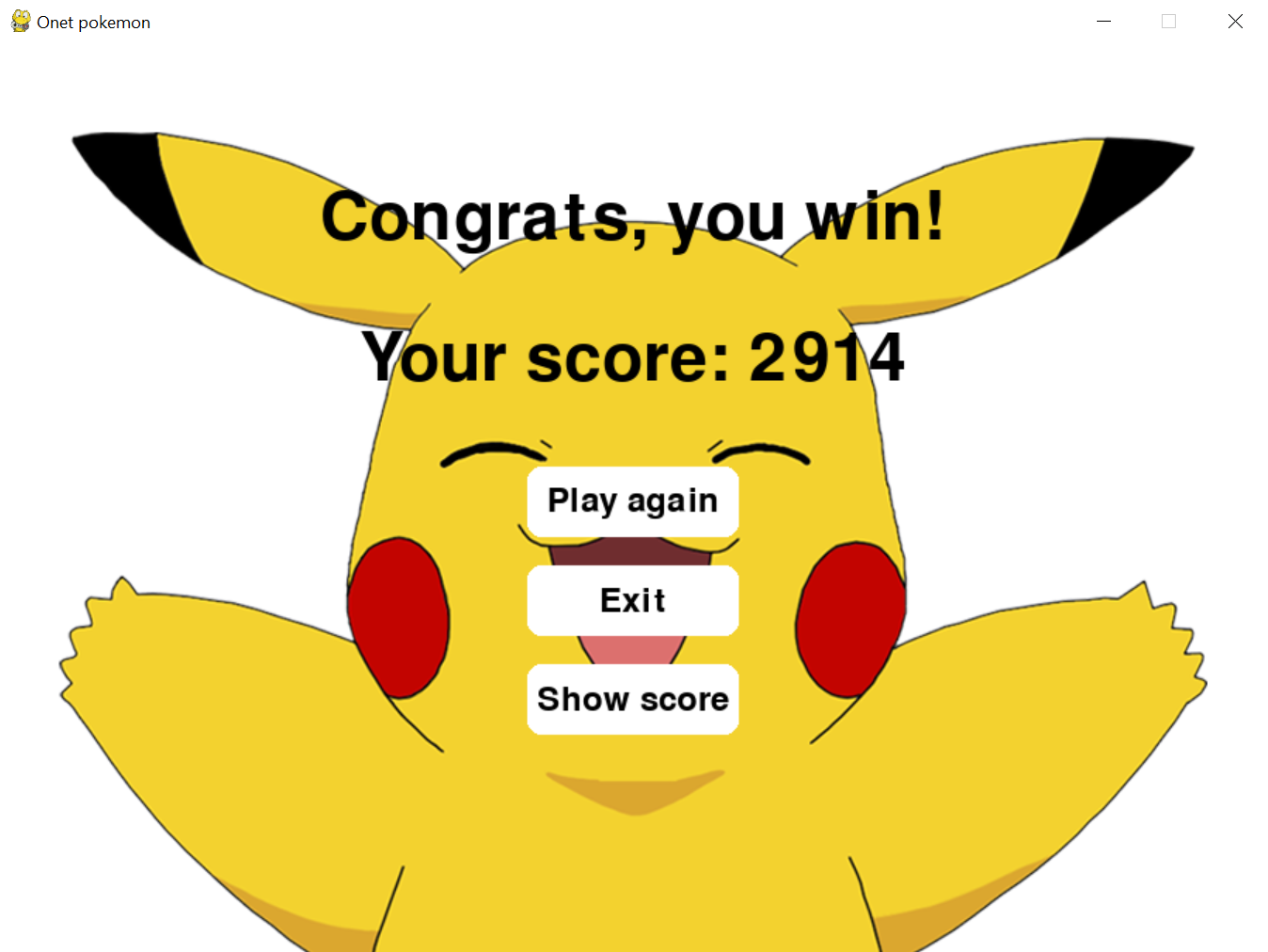
* + 1. Giao diện trang sau mỗi level

A screenshot of a cartoon character

Description automatically generated

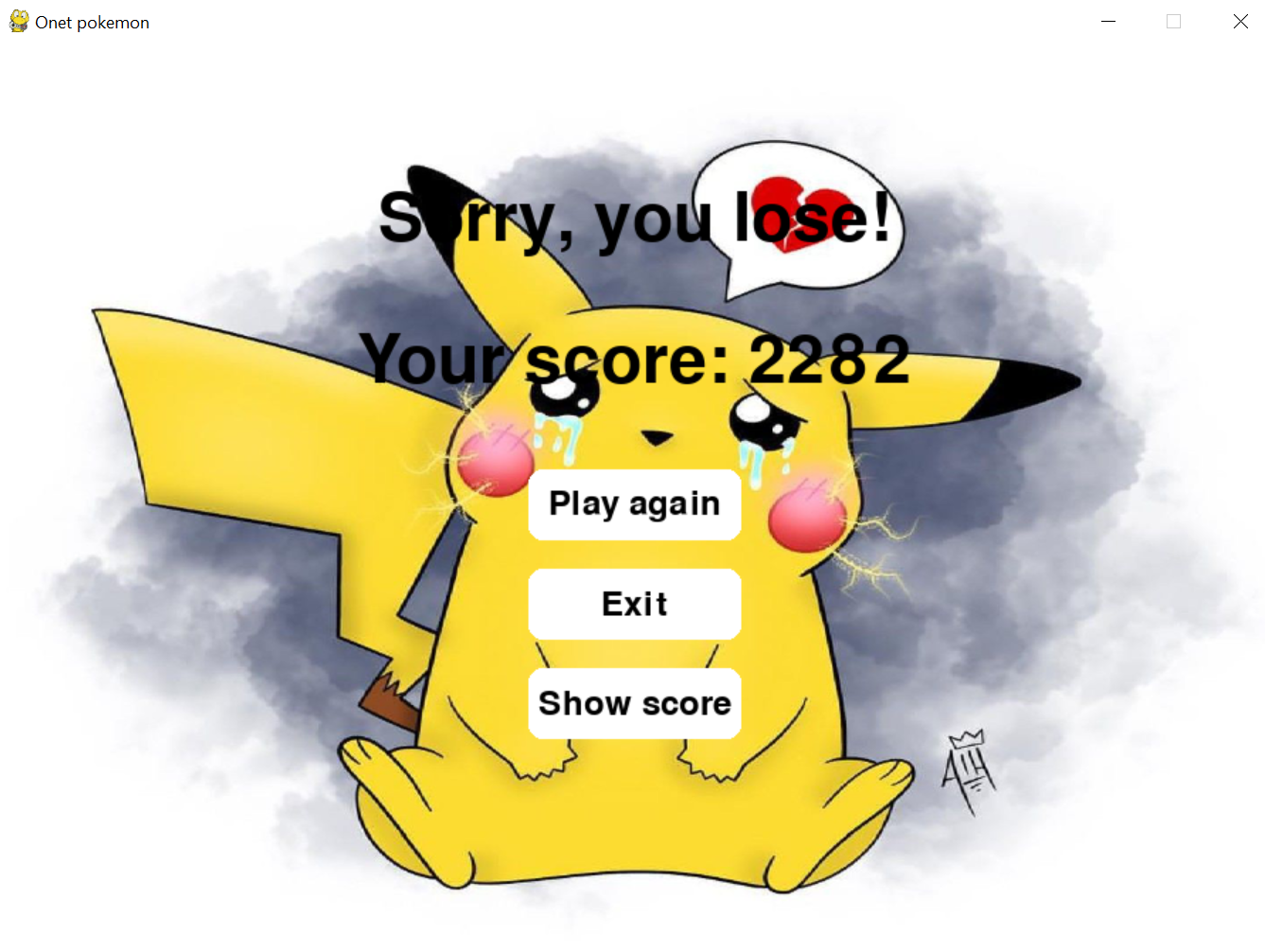
Giao diện màn hình sau mỗi level

* + 1. Giao diện trang chiến thắng



Giao diện màn hình khi chiến thắng

* + 1. Giao diện trang thất bại



Giao diện màn hình khi thất bại

* + 1. Giao diện thống kê điểm



Giao diện màn hình thống kê điểm

**TÀI LIỆU THAM KHẢO**

[1] David J.Malan, *CS50’s Introduction to Programming with Python,* 1/12/2023, https://cs50.harvard.edu/python/2022/

[2] Jon Finder, *Pygame: A Primer on Game Programming in Python*, 1/12/2023, https://realpython.com/pygame-a-primer/

[3] Nguyễn Văn Hiếu (2023), *Hàng đợi – Queue,* 12/12/2023, https://nguyenvanhieu.vn/hang-doi-queue/