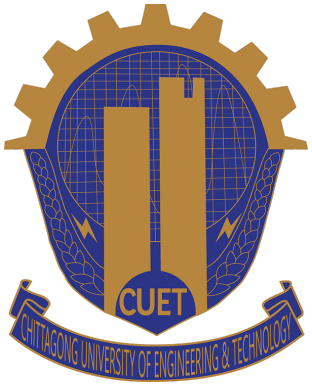
Chittagong University of Engineering and Technology



Slide Puzzle Game Using Hand Gestures

Shariful Islam (1804011)

Md. Sajjadu Zaman (1804016)
Kawsar Ahmed Shamim (1804017)

February 2023

Department of Computer Science and Engineering

Abstract

The slide puzzle game is a unique gaming experience that uses hand gestures for control, developed using the powerful combination of Python, PyGame, and OpenCV. This game offers players a fun and interactive way to solve puzzles by sliding the puzzle pieces with their hand gestures. The game uses OpenCV to track hand movements and translate them into actions in the game, making it a highly immersive and engaging experience. The game is built using PyGame, a popular game development library in Python, which provides a smooth and seamless experience for players. The slide puzzle game features beautiful graphics and a user-friendly interface, making it easy for players of all ages and skill levels to pick up and play. The game offers a range of challenging levels that will keep players engaged for hours on end. With its unique hand gesture control system, the slide puzzle game is sure to be a hit with players who are looking for a new and exciting way to enjoy puzzle games. Whether you're a casual or hardcore gamer, this game is sure to provide an enjoyable experience for everyone.

Contents

Chapter 1

Introduction

Slide puzzle games have been around for decades and are a staple in the gaming world. They offer a challenging and engaging gameplay experience that requires players to use their problem-solving skills to rearrange puzzle pieces and form a complete picture. The latest evolution in slide puzzle games is the use of hand gestures to control the game, rather than traditional mouse and keyboard controls. This new development takes the classic gameplay experience to the next level, making it more immersive and interactive for players.

This new slide puzzle game is developed using Python, PyGame, and OpenCV. Python is a popular and versatile programming language that is used for a wide range of applications, including gaming. PyGame is a library for Python that is specifically designed for game development, providing a range of tools and functions that make it easier to create games. OpenCV, on the other hand, is a computer vision library that is used for real-time image processing and object recognition. Together, these technologies provide the foundation for this innovative slide puzzle game that uses hand gestures to control the game.

In the game, players are presented with a puzzle board containing several pieces that they must rearrange to form a complete picture. The hand gesture recognition system uses the camera on the player's device to detect and track number of fingers which are up, changing the number of fingers up the players can move the puzzle pieces around the board, either up-down or right-left, allowing players to interact with the game in a more natural and intuitive way. The game provides real-time feedback on the player's hand gestures, making it easy to see if the movement was successful or not.

Chapter 2

Methodology

We used a bunch of functions and methods to make the project a succesful one. In this chapter we will be discusing about those functions and methos.

2.1 Detecting Fingers

The function, `NoOfFingers()`, captures video from the default camera (0) and detects hands in the video using an instance of the `HandDetector` class. The `HandDetector` class is responsible for detecting hands in an image using computer vision techniques. The function reads a single frame of video using `cv2.VideoCapture` and passes it to the `HandDetector` instance's `findHands` method, which returns an image with the detected hands and a list of hand regions in the image. The function then counts the number of fingers raised in the first hand detected using the `fingersUp` method. Finally, the function returns the number of fingers and the number of hands detected as a tuple. If no hands are detected, the function returns (0, 0).

2.2 The Main Game Loop

In the main game loop, the `slideTo` variable will track which direction the player wants to slide a tile (it starts off at the beginning of the game loop as None and is set later) and the `msg` variable tracks what string to display at the top of the window. The program does a quick check if the board data structure has the same value as the solved board data structure stored in `SOLVEDBOARD`. If so, then the `msg` variable is changed to the string 'Solved!'. This won't appear on the screen until `drawBoard()` has been called to draw it to the `DISPLAYSURF` Surface

object and `pygame.display.update()` is called to draw the display Surface object on the actual computer screen.

2.2.1 Handling Events

Before going into the event loop, the program calls `checkForQuit()` to see if any QUIT events have been created (and terminates the program if there have). Why we have a separate function (the `checkForQuit()` function) for handling the QUIT events will be explained later. The for loop executes the event handling code for any other event created since the last time `pygame.event.get()` was called (or since the program started, if `pygame.event.get()` has never been called before). If the type of event was a `MOUSEBUTTONUP` event (that is, the player had released a mouse button somewhere over the window), then we pass the mouse coordinates to our `getSpotClicked()` function which will return the board coordinates of the spot on the board the mouse release happened. The `event.pos[0]` is the X coordinate and `event.pos[1]` is the Y coordinate. If the mouse button release did not happen over one of the spaces on the board (but obviously still happened somewhere on the window, since a `MOUSEBUTTONUP` event was created), then `getSpotClicked()` will return None. If this is the case, we want to do an additional check to see if the player might have clicked on the Reset, New, or Solve buttons (which are not located on the board).

The coordinates of where these buttons are on the window are stored in the `pygame.Rect` objects. We can pass the mouse coordinates from the Event object to the `collidepoint()` method. This method will return True if the mouse coordinates are within the `Rect` object's area and False otherwise.

Checking QUIT Events

The `checkForQuit()` function will check for QUIT events (or if the user has pressed the `Esc` key) and then call the `terminate()` function. But this is a bit tricky and requires some explanation. PyGame internally has its own list data structure that it creates and appends Event objects to as they are made. This data structure is called the event queue. When the `pygame.event.get()` function is called with no parameters, the entire list is returned. However, you can pass a constant like `QUIT` to `pygame.event.get()` so that it will only return the `QUIT` events (if any) that are in the internal event queue. The rest of the events will stay in the event queue for the next time `pygame.event.get()` is called.

Sliding Tiles with the Mouse

If `getSpotClicked()` did not return (None, None), then it will have returned a tuple of two integer values that represent the X and Y coordinate of the spot on the board that was clicked. Then the if and `elif` statements check if the spot that was clicked is a tile that is next to the blank spot (otherwise the tile will have no place to slide). Our `getBlankPosition()` function will take the board data structure and return the X and Y board coordinates of the blank spot, which we store in the variables `blankx` and `blanky`. If the spot the user clicked on was next to the blank space, we set the `slideTo` variable with the value that the tile should slide.

Sliding Tiles with the Keyboard

We can also let the user slide tiles by pressing keyboard keys. The `if` and `elif` statements let the user set the `slideTo` variable by either pressing the arrow keys or the WASD keys (explained later). Each `if` and `elif` statement also has a call to visValidMove() to make sure that the tile can slide in that direction. (We didn't have to make this call with the mouse clicks because the checks for the neighboring blank space did the same thing.)

Sliding Tiles with Hand Gestures

After the event loop is finished the program checks whether the game mode is now `HAND` mode. If so then `FingerDetector.NoOfFingers()` is called and number of fingers, hands are calculated. If the hands is not zero then the program checks appropriate number of fingers for appropriate move. If the `fingers == 1` then `slideTo = UP`, `if fingers == 2` then `slideTo = DOWN`, `if fingers == 3` then `slideTo = RIGHT, if fingers == 4` then `slideTo = LEFT` is assigned and also checks whether the move is valid or not by calling `isValidMove(mainBoard, MOVE)` function which checks whether the move position is blank or not. If it is blank then the move is valid and `isValidMove(mainBoard, MOVE)` returns `True` otherwise returns `False`.

2.3 Creating the Board Data Structure

The `getStartingBoard()` data structure will create and return a data structure that represents a —solved board, where all the numbered tiles are in order and the blank tile is in the lower right corner. This is done with nested for loops, just like the board data structure in the Memory Puzzle game was made. However, notice that the first column isn't going to be [1, 2, 3] but

instead [1, 4, 7]. This is because the numbers on the tiles increase by 1 going across the row, not down the column. Going down the column, the numbers increase by the size of the board's width (which is stored in the `BOARDWIDTH` constant). We will use the counter variable to keep track of the number that should go on the next tile. When the numbering of the tiles in the column is finished, then we need to set counter to the number at the start of the next column.

2.3.1 Drawing the board

This function handles drawing the entire board and all of its tiles to the `DISPLAYSURF` display Surface object. The fill() method completely paints over anything that used to be drawn on the display Surface object before so that we start from scratch. Then the program handles drawing the message at the top of the window. We use this for the —"Generating new puzzle..." and other text we want to display at the top of the window. Remember that if statement conditions consider the blank string to be a False value, so if message is set to " then the condition is False. Next, nested for loops are used to draw each tile to the display Surface object by calling the `drawTile()` function.

Drawing a Tile

The `drawTile()` function will draw a single numbered tile on the board. The `tilex` and `tiley` parameters are the board coordinates of the tile. The number parameter is a string of the tile's number (like '3' or '12'). The `adjx` and `adjy` keyword parameters are for making minor adjustments to the position of the tile. For example, passing 5 for `adjx` would make the tile appear 5 pixels to the right of the `tilex` and `tiley` space on the board. Passing -10 for `adjx` would make the tile appear 10 pixels to the left of the space. These adjustment values will be handy when we need to draw the tile in the middle of sliding. If no values are passed for these arguments when `drawTile()` is called, then by default they are set to 0. This means they will be exactly on the board space given by `tilex` and `tiley`. The PyGame drawing functions only use pixel coordinates, this program converts the board coordinates in `tilex` and `tiley` to pixel coordinates, which we will store in variables left and top (since `getLeftTopOfTile()` returns the top left corner's coordinates). We draw the background square of the tile with a call to `pygame.draw.rect()` while adding the `adjx` and `adjy` values to left and top in case the code needs to adjust the position of the tile. Then creating the Surface object that has the number text drawn on it. A `Rect` object for the Surface object is positioned, and then

used to `blit` the Surface object to the display Surface. The `drawTile()` function doesn't call `pygame.display.update()` function, since the caller of `drawTile()` probably will want to draw more tiles for the rest of the board before making them appear on the screen.

Making Text Appear on the Screen

The `makeText()` function handles creating the Surface and `Rect` objects for positioning text on the screen. Instead of doing all these calls each time we want to make text on the screen, we can just call `makeText()` instead. This saves us on the amount of typing we have to do for our program. (Though `drawTile()` makes the calls to render() and $get_rect()$ itself because it positions the text Surface object by the center point rather than the `topleft` point and uses a transparent background color.)

2.3.2 Creating a New Puzzle

The `generateNewPuzzle()` function will be called at the start of each new game. It will create a new board data structure by calling `getStartingBoard()` and then randomly scramble it. The first few lines of `generateNewPuzzle()` get the board and then draw it to the screen (freezing for half a second to let the player see the fresh board for a moment). The `numSlides` parameter will show tell the function how many of these random moves to make. The code for doing a random move is the `getRandomMove()` call to get the move itself, then call `slideAnimation()` to perform the animation on the screen. Because doing the slide animation does not actually update the board data structure, we update the board by calling `makeMove()`. We need to keep track of each of the random moves that was made so that the player can click the —Solve button later and have the program undo all these random moves. So the move is appended to the list of moves in sequence. Then we store the random move in a variable called `lastMove` which will be passed to `getRandomMove()` on the next iteration. This prevents the next random move from undoing the random move we just performed. All of this needs to happen `numSlides` number of times, so we put `getRandomMove()` inside a for loop. When the board is done being scrambled, then we return the board data structure and also the list of the random moves made on it.

Animating the Tile Slides

The first thing our tile sliding animation code needs to calculate is where the blank space is and where the moving tile is. The code that calls `slideAnimation()` should make sure that the slide it passes for the direction parameter is a valid move to make. In order to draw the frames of the sliding animation, we must draw the `baseSurf` surface on the display Surface, then on each frame of the animation draw the sliding tile closer and closer to its final position where the original blank space was. The space between two adjacent tiles is the same size as a single tile, which we have stored in `TILESIZE`. The code uses a for loop to go from 0 to `TILESIZE`. Normally this would mean that we would draw the tile 0 pixels over, then on the next frame draw the tile 1 pixel over, then 2 pixels, then 3, and so on. Each of these frames would take `1/30th` of a second. If you have `TILESIZE` set to 80 (as the program in this book does on line 12) then sliding a tile would take over two and a half seconds, which is actually kind of slow. So instead we will have the for loop iterate from 0 to `TILESIZE` by several pixels each frame. The number of pixels it jumps over is stored in `animationSpeed`, which is passed in when `slideAnimation()` is called. For example, if `animationSpeed` was set to 8 and the constant `TILESIZE` was set to 80, then the for loop and `range(0, TILESIZE, animationSpeed)` would set the i variable to the values 0, 8, 16, 24, 32, 40, 48, 56, 64, 72. (It does not include 80 because the range() function goes up to, but not including, the second argument.) This means the entire sliding animation would be done in 10 frames, which would mean it is done in `10/30th` of a second (a third of a second) since the game runs at 30 FPS.

Getting Random Moves

At the beginning of the game, we start with the board data structure in the solved, ordered state and create the puzzle by randomly sliding around tiles. To decide which of the four directions we should slide, we'll call our `getRandomMove()` function. Normally we could just use the random.choice() function and pass it a tuple (UP, DOWN, LEFT, RIGHT) to have Python simply randomly choose a direction value for us. But the Sliding Puzzle game has a small restriction that prevents us from choosing a purely random number. If you had a slide puzzle and slid a tile to left, and then slid a tile to the right, you would end up with the exact same board you had at the start. It's pointless to make a slide followed by the opposite slide. Also, if the blank space is in the lower right corner than it is impossible to slide a tile up or to the left. The code in `getRandomMove()` will take these factors into account. To prevent

the function from selecting the last move that was made, the caller of the function can pass a directional value for the `lastMove` parameter. Then wetake a list of all four directional values stored in the `validMoves` variable. The `lastMove` value (if not set to None) is removed from `validMoves`. Depending on if the blank space is at the edge of the board,then we remove other directional values from the `lastMove` list. Of the values that are left in `lastMove`, one of them is randomly selected with a call to random.choice() and returned.

2.3.3 Animating the Board on "Reset" or "Solve"

When the player clicks on —Reset or —Solve, the Slide Puzzle game program needs to undo all of the moves that were made to the board. The list of directional values for the slides will be passed as the argument for the `allMoves` parameter. We use list slicing to create a duplicate of the `allMoves` list. Remember that if you don't specify a number before the :, then Python assumes the slice should start from the very beginning of the list. And if you don't specify a number after the :, then Python assumes the slice should keep going to the very end of the list. So `allMoves[:]` creates a list slice of the entire `allMoves` list. This makes a copy of the actual list to store in `revAllMoves`, rather than just a copy of the list reference. To undo all the moves in `allMoves`, we need to perform the opposite move of the moves in `allMoves`, and in reverse order. There is a list method called reverse() which will reverse the order of the items in a list. We call this on the `revAllMoves` list on line 316. Then a for loop iterates over the list of directional values. Then we call `slideAnimation()` to perform the animation, and `makeMove()` to update the board data structure.

2.4 The Auto Solve Algorithm

Solving a slide puzzle can be really tricky. We could program the computer to do it, but that would require us to figure out an algorithm that can solve the slide puzzle. That would be very difficult and involve a lot of cleverness and effort to put into this program. Fortunately, there's an easier way. We could just have the computer memorize all the random slides it made when it created the board data structure, and then the board can be solved just by performing the opposite slide. Since the board originally started in the solved state, undoing all the slides would return it to the solved state. Suppose, after the right slide, if we do the opposite slide (a left slide) then the board will be back in the original state. So to get back to the original state

after several slides, we just have to do the opposite slides in reverse order. If we did a right slide, then another right slide, then a down slide, we would have to do an up slide, left slide, and left slide to undo those first three slides. This is much easier than writing a function that can solve these puzzles simply by looking at the current state of them.

Chapter 3

Tools and Technologies

The development of a slide puzzle game using hand gestures is a unique and innovative approach to traditional gaming. The game is created using a combination of Python, Pygame, and OpenCV, which are powerful tools for creating interactive and visually engaging applications. The use of hand gestures, instead of traditional keyboard or mouse controls, adds an extra layer of interactivity and immersion to the game, making it an exciting and unique experience for players. Pygame is a popular library for game development, providing a simple and efficient way to create games with powerful graphics and sound. OpenCV, on the other hand, is a computer vision library that provides tools for image and video processing, allowing the game to detect and respond to hand gestures in real-time. Python provides the underlying structure and logic for the game, making it a flexible and scalable platform for development.

3.1 Python

Python is a high-level programming language that is widely used for various applications such as web development, scientific computing, data analysis, and artificial intelligence. It is known for its simplicity, readability, and versatility, making it an excellent choice for both novice and experienced programmers. Python was first released in 1991 and has since become one of the most popular programming languages in the world, with a large and supportive community of developers.

One of the key features of Python is its dynamically-typed nature, which allows developers to focus on the logic of their code rather than worrying about the underlying data types. Python

also has a large standard library, which provides a wide range of modules and packages for tasks such as connecting to the web, reading and writing files, and data analysis. This makes it easy for developers to get started with programming and to build complex applications quickly and efficiently.

Python is also known for its support for multiple programming paradigms, including procedural, object-oriented, and functional programming. This makes it a versatile language that can be used for a wide range of applications, from simple scripts to complex web applications.

In recent years, Python has become increasingly popular in the fields of data science and machine learning, with its libraries such as NumPy, Pandas, and Matplotlib providing powerful tools for data analysis and visualization. With its ease of use, extensive libraries, and growing popularity, Python is a language that is well-positioned for continued growth and success in the future.

3.2 PyGame

Pygame is a popular and widely used library for game development in Python. It was created to provide a simple and easy-to-use set of tools for game development, making it an accessible option for both beginner and experienced game developers. Pygame provides a wide range of features for game development, including support for 2D graphics, sound, and animation, as well as user input and event handling.

One of the strengths of Pygame is its simplicity, which makes it easy for developers to get started with game development. The library provides a high-level interface to various system functions, such as graphics and sound, making it easier for developers to focus on the game logic and design. Pygame also has a large and active community of users, who contribute to the development of the library and provide support for other users.

Pygame is also highly portable and can run on a wide range of platforms, including Windows, macOS, and Linux, as well as various handheld devices and gaming consoles. This makes it a versatile option for game development, as developers can write their games once and run them on multiple platforms with minimal changes.

In addition to its ease of use, Pygame also provides powerful tools for game development, such as support for various image and sound formats, and a wide range of graphics and animation features. These features, combined with the large and growing library of user-contributed

modules, make Pygame a powerful and versatile tool for game development.

Whether you're a beginner or an experienced game developer, Pygame provides an accessible and powerful platform for game development in Python. With its simple interface, extensive libraries, and large community of users, Pygame is an excellent choice for anyone looking to develop games in Python.

3.3 OpenCV

OpenCV is an open-source computer vision library that provides a comprehensive set of tools for image and video processing. It was created to provide a common platform for computer vision and machine learning algorithms, making it an essential tool for a wide range of applications, from simple image processing to complex computer vision systems. OpenCV was first released in 1999 and has since become one of the most widely used computer vision libraries, with a large and active community of users and developers.

OpenCV provides a wide range of features for image and video processing, including support for image and video capture, image processing, object detection, and machine learning algorithms. It also provides a set of tools for implementing computer vision algorithms, including feature detection, object recognition, and motion estimation. These tools are highly optimized, making them fast and efficient, even for large and complex images and videos.

In addition to its versatility, OpenCV also provides a set of powerful machine learning algorithms for object recognition and image classification. These algorithms can be used for a wide range of applications, from simple image classification to complex object recognition systems. The library also provides a set of tools for training machine learning models, making it easy for developers to create custom models for their specific needs.

Overall, OpenCV is a comprehensive and powerful library for computer vision and machine learning. With its wide range of features, ease of use, and growing community of users and developers, OpenCV is an essential tool for anyone looking to develop computer vision applications. Whether you're a beginner or an experienced computer vision developer, OpenCV provides the tools and resources you need to create powerful and effective computer vision systems.

Chapter 4

Result

The development of a slide puzzle game using hand gestures, built with Python, Pygame, and OpenCV, has produced impressive results. This innovative solution provides a new and engaging way for players to control and solve slide puzzles. By using hand gestures, players can easily move the puzzle pieces around the game board, eliminating the need for traditional input methods such as a keyboard or mouse.

The use of Python and Pygame allowed for the creation of a smooth and responsive game environment, while OpenCV provided the ability to recognize hand gestures with high accuracy. The hand gesture recognition system was integrated seamlessly into the game, making the experience intuitive and effortless for players.

In addition to being a fun and engaging way to solve slide puzzles, this tool has the potential to be used in various other applications. For example, it could be used as an educational tool in schools or as a therapeutic tool for individuals with physical disabilities. The development of this tool showcases the power of combining the latest technology with creative problem solving, and the results are truly impressive.

In conclusion, the development of a slide puzzle game using hand gestures has produced a unique and innovative solution that provides players with an enjoyable and engaging gaming experience. With the combination of Python, Pygame, and OpenCV, this tool demonstrates the potential for new and innovative forms of interaction in gaming and other applications.

Chapter 5

Conclusion

In conclusion, the development of a slide puzzle game using hand gestures has been a successful project that showcases the potential of cutting-edge technologies such as Python, Pygame, and OpenCV. The integration of hand gestures as the primary control mechanism has provided a unique and innovative experience for players, making the game both fun and interactive. The use of Python and Pygame has allowed for a seamless and smooth gaming experience, while OpenCV has provided the ability to accurately track hand movements and convert them into in-game actions. The result is a highly engaging game that offers a new level of interaction, making the game both entertaining and challenging. This project has demonstrated the potential of combining innovative technologies and game design to create a new level of immersive gaming experiences. The future of gaming is exciting and this project is a testament to the potential for new and innovative games to be developed using cutting-edge technologies. Overall, the development of this slide puzzle game using hand gestures has been a great success and has laid the foundation for further advancements in the gaming industry.