Name

Flap and Dash is a 2D game where players control a small character, guiding it through a series of obstacles with just a tap to make the character goes against gravity. The goal is to navigate it through these gaps without hitting the pipes or the ground, scoring points for every successful pass. The longer you survive, the higher your score.

How to demo  
Start the application, click the Start button and your character is going to show up. Then you must press space every time you wish to make it dash against gravity. You are going to see some pipes; you must go through the hole between these obstacles. If you touch any of the pipes you lost, then you can either click on the New Game button or the Exit one. The “New Game” button is rather self-explanatory, as it restarts the game. Also, there’s a score counter on the top of your screen.  
Features:

* **Physics-based Gameplay**: Gameplay mechanics are based on realistic physics, where gravity is a constant force acting on the character, pulling it down towards the ground. Players must tap or click to make it momentary lift upward, creating a delicate balance between ascending and descending. This physics-based challenge makes it crucial for players to time their movements carefully to avoid crashing while navigating through the obstacles.
* **Tap to Fly Mechanics**: Players tap to keep the character in the air, dodging obstacles along the way. We are going to use the KeyListener to detect user taps from the AWT library. There's no lag between taps, so you can keep tapping as much as you need. But, you can only use this during the main game loop, tapping doesn’t work on the start or game over screens (since they’re basically the same thing).
* **Infinite Game Loop**: The game operates in a continuous loop, allowing the player to keep playing indefinitely. This persists until a game-over condition is triggered, which happens when the player's character either collides with one of the obstacles, such as a pipe, or crashes into the ground. There are no fixed levels or end points, and the game resets upon failure, encouraging players to improve their score each time they play.
* **Randomly Generated Pipes**: We’re using math.random() from the java.lang library to generate the pipes at random heights, adding to the challenge. The gaps between the pipes stay consistent both vertically and horizontally, keeping the gameplay balanced while still unpredictable.
* **Gravity Flipping Gates**: We are also using math.random() from the java.lang library to generate every 5-15 pipes a gate, and when the object passes through, the direction of the gravity gets flipped. Furthermore, whenever you go through another gate it will be back to normal. This feature makes our game authentic and unique.
* **Scoring System:** We are going to use a JLabel from the Swing library. The game features a simple yet rewarding scoring system, where players accumulate points for every successful pass through a pair of pipes. Each time the character safely navigates between the obstacles without colliding, the score increases by one. The goal is to achieve the highest score possible.
* **Game Over Screen and Restart Option:** We are going to use the JButtons from the Swing library. After the character collides with a pipe or hits the ground, the game transitions to a game over screen. This screen displays the player's final score. From here, players are given the option to quickly restart the game with a single click, enabling seamless and immediate replays. In case the player wishes to quit you also have a button for that. This fast-paced restart feature is designed to maintain the player's engagement.

Topics of choice

* **Physics Simulation**: In game development, simulating basic physics is crucial to make gameplay feel smooth and natural. Gravity basically pulls the character down at a constant rate, so every frame its position needs to be updated based on that. When the player taps to jump, you apply an upward force that temporarily counters the gravity, letting it rise for a moment before it’s pulled down again. If gravity’s too weak or the jump is too high, the game becomes too easy, but if they’re too strong, it gets frustratingly hard. It’s all about fine-tuning those values to create a good flow.
* **Collision Detection**: This is how you know when it crashes into a pipe or hits the ground. Essentially, you need to track if the character’s hitbox (basically, an invisible rectangle around it) overlaps with any of the pipes or the ground. The easiest way to do this is with axis-aligned bounding boxes, which is just a fancy way of saying you check if two rectangles are touching. This method is pretty efficient and works great for a game like this. When a collision happens, that’s when you’d trigger a game over or another event. It’s important to get this working smoothly since it directly affects how the game feels. For example, if the collision detection is off, players will notice and get frustrated.
* **Procedural Content Generation**: Procedural generation is what keeps each playthrough unique and challenging. Instead of pre-designing where every pipe goes, you write some code to randomly generate them as the game goes on. This means that no two runs are the same, keeping players entertained. The key here is controlling the randomness so the game stays fun but fair—like making sure the gaps between the pipes are doable and not too tight or too wide. This dynamic generation keeps things interesting without overwhelming the player right from the start.