README

Doodle Jump
Charlotte Le (3035718399) & Gautam Anand (3035906938)
TA: Kathleen Gao

Intro
This is a modified version of Doodle Jump. The player will be released from the top of the screen and continuously be pulled down. If the player falls to the bottom of the screen, then it is game over. However, the player can prevent this by landing on top of platforms to jump. Every time a player lands on a platform, they will bounce, and their score increases. These platforms will randomly come from the left side of the screen and move to the right of the screen.

Main Features

1. The first feature is the player's movement. The player is able to smoothly move left and right. Moreover, the player is continuously pulled downwards. However, if the player collides with a platform, the player will bounce up once (this was a feature in our original proposal).
2. As the game progresses, there are random platforms that approach from the left side of the screen and move to the right of the screen before disappearing (this was a feature in our original proposal, but it is slightly modified; we originally proposed to have the platforms move from top to bottom, but we changed this to left to right to make the game more original).
3. There is a score tracker that increases every time the player collides with the platform. This score appears larger at the end of the game so that the player can easily identify the score.

Extra Features

1. The player can click the "play" button to start the game. Moreover, once the game has ended, the player is able to click the "play again" button to return to the home screen so that they are able to play the game again. Additionally, there is a "readme" button on the home screen so that players can access this file to learn more about the game (NOTE: we talked to Lam during office hours and concluding that the command used to open the PDF file only worked on specific operating systems. For some operating systems, nothing will happen. We decided to keep it since it is an extra feature that we incorporated just for fun/to learn!).
2. If the player moves too far right, the player will appear on the left side of the screen. Similarly, if the player moves too far left, the player will appear on the right side of the screen. If the player moves too far below, then the "game over" page appears.

Justification for Complexity

1. How did your features contribute to complexity?

   Our project is a fully operational product and can be played by any user on any laptop (minus opening the "readme" file. The finished product is neither too complex, nor too simple (but surely something we are proud of!). In order to perform seemingly complex algorithms like collision detection (between the player and the platform), we did not make use of any built-in functions; instead, we came up with a unique implementation based on the distance between the player and the platform. This was calculated using the coordinate geometry formula in mathematics. We also allowed the user to play the game for as long as they want to, by clicking the 'play again' button. Additionally, we employed border checking methods in both the x and y directions. If the player moves too

far right, the player reappears on the left side of the screen. If the player moves too far left, the player reappears on the right side of the screen. If the player goes below the screen, the game is over and the final score of the player is shown. Overall, our implementations of the algorithms are completely unique and represent original thought.

2. Did you make use of more advanced features of Python?
   We made use of various libraries such as "pygame", "math", "random", "subprocess", etc. Learning about how these work and the various functions/methods they had to complete the final product was very satisfying (albeit time-consuming)!
3. How did you break up complicated tasks?
   Throughout the project, we divided all the large operations into smaller tasks, effectively making use of the style of programming known as functional programming. For example, we wrote separate functions for the movement of the player, the generation of platforms, the collision detection between the player and the platforms, and restarting the game. At the end of the code, we call the run function, which cleanly puts everything together and makes our game work.

Lists

- There are two lists in our project: the "self.platform_x" and "self.platform_y" lists.
- These lists store the x and y coordinates of newly generated platforms.
- The lists are constantly updated so that there are only "num_platforms" number of items in a list at one time.
- The lists are empty at the beginning of the game, but once the game starts, platforms are generated so items are added to the lists.
- There is a random number generator within a range to ensure that the platforms are created randomly (i.e. items are added to the list randomly)
- Since these lists are very similar, we have added multiple additional functions to make up for the fact that the second list is not that different from the first list.

Local Variables

- We have used many local variables inside functions.
- In our initialization method, we have almost 20 instance attributes.
- Some of these variables, such as those that store images, are not modified.
- However, other variables, such as the player's position, are continuously modified.
- Moreover, we used variables in our other methods, such as the "distance" variable which calculates the distance between the player and any platform to trigger collision.
- There is a variable called "start" which determines whether the game has started or not.

Function Table

Block / Function Name	**Domain (inputs)**	**Range (outputs)**	**Behavior (role in the context of the project)**
__init__	"self" argument	No outputs.	Declares instance attributes such as height and width of screen, player positions, platform positions, etc.
platform	"self" argument	No outputs.	Creates random x and y positions for the platforms and moves the platforms across the screen from left to right.

movement	"self" argument	No outputs.	Controls the movement of the player in both the x and y directions. Runs the appropriate code if the player moves too far left, right, or bottom. Displays the "game over" screen when the player falls down.
collision	"self" argument	Returns a Boolean value; True when collision takes place, False otherwise	Checks whether the player collides with the platform by calculating the distance between the player and the platform.
run	"self" argument	No outputs.	Displays the home screen and allows the user to start the game. This function contains the main game loop, and all the other functions of the class are called inside "run".
changescore	"self" argument	No outputs.	Makes the player jump and adds points to the player's score each time a collision takes place.
restart	"self" argument	No outputs.	Allows the player to play the game again if desired.

Extra Credit

- There were many features of Python that CS10 did not explicitly teach in our project.
- Although we have worked with Pygames in our last project, for this project, we took the time to understand in depth how Pygames works. This was not taught in the class and we have little to no understanding of using Pygames before starting this project.
- Through Pygames, we learned how to:
    - Set the screen up
    - Name the game
    - Change the icon
    - Determine a font
    - Access and load pictures
    - Update the score on the screen
    - Display the game
    - Move the player left and right
    - Respond to mouse clicks in certain positions
    - Move the player more smoothly with the "clock" variable
    - Integrate events
    - Exit the game
- Moreover, we learned how to open PDFs through the "subprocess" module.
- Our "collision" function is a new idea that was not taught in class. We had to use our knowledge of the distance equation in mathematics for this function.
- Finally, we learned Photoshop techniques to edit the images so that they look better and work with Pygames (e.g. resizing images and designing the "home" + "game over" screen)!