# Lucky 7s

## **Main Aspects of the game**

1. Credit Meter: It indicates the total amount available for playing. For testing purposes, if the amount goes below 50 credits, it will automatically add some more.
2. Bet Meter: It indicates the bet of the current game. This amount will be deducted from the credit meter once the user plays the game.
3. Change Bet Button: Possible bets are from 10 to 100 with increments of 10. Pressing this button goes to the next possible bet.
   1. I have manually entered the possible bets instead of auto-incrementing by 10 because of its flexibility. Suppose we want to modify the possible bets to 10,20,25,50,75,100. We will have to update only in one place instead of changing the logic.
   2. Ideally, it should be handled in a separate math file but this implementation is pretty straightforward
4. Max Bet Button: Put the user directly to the last possible bet. It is not hardcoded to allow custom bet options.
5. Exit Game Button: When running the application, it allows the player to close it.

## **Flow and Functionalities:**

1. When the user rolls the dice, all the buttons are dimmed (disabled) to show that these are not active. It becomes active after reaching the idle state.
   1. Exit dim-icon was not available and thus it is not dimmed. (need access to graphics)
2. State Change: I felt 2 states are enough to correctly represent the game instead of the mentioned 4. It allowed me to have a clear distinction in the functionality of each one.
   1. **Idle state**: The game is loaded to this state. In this state, users have different input options. Users can set up the bet configuration and play the game.
   2. The game is OFFICIALLY started only when credit is deducted. Therefore, we wait for credit to be deducted before transitioning to the Playing state.
   3. **Playing state**: It signifies that the game is in motion and the user does not have control over the game's flow.
   4. First, we predetermine the output, and then depending on it, values on both the dices are determined randomly.
   5. **In order to simulate the dice roll, I animated the numbers on the dice randomly, and the border color changed to dark red. Once the simulation part is done, the border is back to the original color, and predefined values are set on the dice.**
   6. Once the gameplay is finished, we transition to the Award state to ensure that the game internals are reflected accordingly and the user is notified.
   7. **Award State**: The user is shown a splash of Win/Lose based on the outcome. After that, the win amount (if any) is reflected in the credit meter. I want to create anticipation of the win being added slowly into the meter, instead of updating the meter directly. It improves the player's involvement.
   8. **If you notice, there is a lag while moving from one state to another. It is intentional. Since the game is small and states are not so distinct, the added pause for each state provides clear evidence of the change in state. Ideally, the state change is dependent on any specific event, like the end of the animation or the completion of the transaction. However, I do not have access to the prebuilt animations or movies that I can work with. Having a fixed time of 2 sec allowed me to replicate a similar situation.**
   9. Best way to view this is through the Unity instead of executable. The logs were added to provide a clear indication of the state change.

## **Self-Assessment:**

1. Due to a lack of familiarity with Unity, the screen resolution is not fixed, and thus can see some extra stuff. This could have been addressed.
2. Lack of access to SPECIFIC and RELATED game assets like images, music, animation, and movies, the game UI is kept very basic. More animations could have been added.
3. I realized halfway through that the game was being developed on a very old version, but again my lack of familiarity prohibited me from upgrading it as it could have cost a significant amount of time.
4. The old version prevented me from implementing more seamless animation using **Dotween**. It could have added a better effect instead of scaling splash up and down.
5. All/most Handlers could be singleton objects.
6. The game is not truly random, nor intended to be. I have fixed the outcome of the game to simulate the **fixed seed random number** generator that is essential for the testing of the game.

## **Improvement scope:**

1. **Understand and make use of serialized fields.**
2. Need to think more about optimization - <https://docs.unity3d.com/2018.4/Documentation/Manual/OptimizingGraphicsPerformance.html>
3. Learn about the coding practices specific to C# like naming conventions.
4. I have used only one Scene for the entire game. More screens could have been added but also increased the complexity. It prevented me from implementing the **CASH-IN functionality**.
5. Each functionality should have 3 aspects **Handler-Logic-Graphics.** Since this game logic is very small, it is integrated in the handler for most cases and the graphical aspect is handled by a separate script.
6. I could have utilized the ‘tag’ feature in the inspector and a few grouping options (like for button) that would help streamline the implementation and flow.
7. Assets are not optimized. Different images and vectors bundled together could have significantly improved the game. Make use of **Sprite Atlas**.
8. Instead of having different functions for win and lose splash, it could be achieved with just one by switching the images.

attribution:

<a href="https://www.flaticon.com/free-icons/you-win" title="you win icons">You win icons created by Freepik - Flaticon</a>

<a href="https://www.flaticon.com/free-icons/lose" title="lose icons">Lose icons created by Freepik - Flaticon</a>

<a href="https://www.flaticon.com/free-icons/exit" title="exit icons">Exit icons created by Freepik - Flaticon</a>