|  |
| --- |
| FLAPPY BIRD USING UNITY AND C#  BY-TANISHA DAS |
|  |

[COMPUTER SCIENCE ANDENGINEERING]

[GARGI MEMORIAL INSTITUTE OF TECHNOLOGY]

[4TH YEAR]

|  |  |  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- | --- | --- |
| Unless you’ve been living in a cave or visiting outer space,  you have certainly played [Flappy Bird](http://en.wikipedia.org/wiki/Flappy_Bird), one of the most successful  games for mobile devices. Hence, I decided to give it a shot and  recreate it using Unity. Needless to mention that all graphics and  sounds found in the tutorial are used only for educational  purposes and do not intend to harm any intellectual  property at all circumstances. Code source and the sprite  sheet used was found here  <https://www.youtube.com/watch?v=ihvBiJ1oC9U>  Game code was written in Visual Studio.  So, let’s start  with the small tutorial! This game’s development  has many aspects, we’ll attempt to touch all of them one by one.  First of all, we used Unity’s spritesheet editor to “slice” the spritesheet  and get references to specific sprites    Then, we created a small animation to simulate the flight course of  the bird (this can be easily accomplished if you drag-drop the relevant  sprites into the scene).      Now, let’s take a more detailed look at our scene. We have various  objects made children of our perspective camera and the Flappy bird.    For background we’ll make a quad object , and place it in so that it  Moves continuously.     |  |  | | --- | --- | |  |  |   The Camera has a CameraFollow script attached that gets  a reference to the Flappy bird transform. The Update script  allows the Camera to update its position based on the bird’s position,  slightly moved to the left. It’s customary for infinite runner games to  have the camera not in the center but a bit left, in order to have  more space on the right so that the player encounters incoming  obstacles/blanks/whatever early enough (in our case, the pipes).  The Floor GameObject holds 2 sprites of the floor, has a  BoxCollider 2D and a FloorMove script and is tagged as “Floor”.  When the first floor moves too much to the left, the script makes the  object move to the right, to simulate that the bird is moving to the  right. You may notice while playing that the “fix” of the floor  movement is somewhat clumsy. One could work that one out with  the –3.9f value modification. For a better solution, I attempted  to utilize mainTextureOffset for smoother scrolling, but it is not  supported by the SpriteRenderer. This would require to have  the floor sprite in another file, import it in Unity as a Texture and set  the mainTextureOffset. This can be done but I felt that it was too  much for the purposes of the tutorial. If you have a better solution,  sound off in the comments section!  image_0F17C8CB  Let’s check the PipeColumnPrefab, which is identical to the  PipeColumnPrefab2 with some sprite differences. The prefab  is made by two sprites that are kinematic rigidbodies and a  triggered BoxCollider2D. Both are tagged as “Pipe”. Plus, between  the two pipes there is an empty GameObject with another  BoxCollider2D, which is tagged as “pipeblank”. One can easily  make out that when the bird hits the pipes, the player will lose  whereas when it hits the “pipeblank” the score will increase by 1.  image_2CF64CCC.png        The Score is a GameObject that holds a reference to the  ScoreManagerScript which contains score-handling code.  The Script is provided the sprites for our numbers.  It starts with a single sprite enabled (Units) and two other  Sprites disabled, specifically Tens and Hundreds,  which are activated when the player’s score is larger than  10 and 100, respectively. It also stores reference to our sprite digits.    In order to save performance, the sprite/score calculation is  achieved if and only if the score has changed from the previous  update call. Then, the respective sprites are enabled and the  relevant score is shown.  The IntroGUI GameObject shows the sprites of the start screen  whereas the DeathGUI (which is disabled at the start of the game)  shows the sprites of the death screen.    image_1D53B10A   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8 | void Update()  {  if (transform.localPosition.x < -3.9f)  {  transform.localPosition = new Vector3  (0, transform.localPosition.y, transform.localPosition.z);  }  transform.Translate(-Time.deltaTime, 0, 0);  } |   Onto the biggest script of our Game, the FlappyScript!  In order to move our Bird to the right, we use this method   |  |  | | --- | --- | | 1  2  3  4 | void MoveBirdOnXAxis()  {  transform.position += new Vector3(Time.deltaTime \* XSpeed, 0, 0);  } |   On the released game, one thing to notice is that during gameplay, the bird begins to look downwards as it falls and looks up when jumping. Consequently, the bird has two states, one going up and one going down.   |  |  | | --- | --- | | 1  2  3  4  5  6 | FlappyYAxisTravelState flappyYAxisTravelState;  enum FlappyYAxisTravelState  {  GoingUp, GoingDown  } |   We use the below method to make sure that the  Flappy bird always has the proper rotation on the z axis.   |  |  | | --- | --- | | 1  2  3  4  5  6  7  8  9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24 | private void FixFlappyRotation()  {  if (GetComponent().velocity.y > 0) flappyYAxisTravelState = FlappyYAxisTravelState.GoingUp;  else flappyYAxisTravelState = FlappyYAxisTravelState.GoingDown;  float degreesToAdd = 0;  switch (flappyYAxisTravelState)  {  case FlappyYAxisTravelState.GoingUp:  degreesToAdd = 6 \* RotateUpSpeed;  break;  case FlappyYAxisTravelState.GoingDown:  degreesToAdd = -3 \* RotateDownSpeed;  break;  default:  break;  }  //clamp the values so that -90<rotation<45 \*always\*  birdRotation = new Vector3(0, 0, Mathf.Clamp(birdRotation.z + degreesToAdd, -90, 45));  transform.eulerAngles = birdRotation;  } |   THANK YOU |