Feedback from session with Intel students:

* Aspect ratio of the game is missing, needs to be 16:9 - Done
* First time they open a code, we need to remind them where the scripts are! - Done
* אתחל במקום התחל - done
* Compilation and error - done
* Step 7 isn’t clear enough. I need to say how to assign a parameter to a new variable and that the function AddForce is already available. Public variables are visible in the inspector - done
* How to change the force before the bird dies? Shortcut for pause key? - done
* The bird Flap animation has a bug in the asset - ????
* When adding new script, I need to say to be in the right folder - done
* When doing copy paste from the word document the line numbers are being copied - done
* How to see compile error – in the console - done
* Float numbers with the ending of f – 130.3f ex. - done
* Spent 1.5 hours for half the 1-4 session

Ars:

Flappy Bird AI Training - feedback with Asif 10/Jan/2019

Table of content and teachers timing - done

Move the Intro to AI into the beginning of the slides - done

Can use English notions combined with Hebrew

**From Genetic algorithm to the neutron network - how it connects**

The hands out should include some of the explanations of the technics being used in the game: like the scrolling backgrounds. - done

Bonus content: Prepare for the fast student - doing

Before the 5/6 potion - give an higher level of what we are going to do

In the 6th part: connect the interfaces to the Theory

In the power point presentation: divide to 3 parts: before hands out 1-4, before 5/6 and before 6.

Content:

1. Unity install zip file
2. Resources: contains unitypackages that will be used during the toturial
3. Project: 01-06 contains the initial state of each toturial
4. Powerpoint presentation containing introduction to Unity, Gaming and AI
5. Students hands out of sections 01-06 to be followed on their own
6. Instructor materials with detailed solutions and further explanations

Instructor material slides:

1. Course Introduction
2. Introduction to unity
3. Introduction to gaming – need to get slides from cyberorg
4. 01: Starting a new project in unity, importing materials
5. 02: Controlling the bird – gaming physics. Adding columns
6. 03: Gamelogic – score, game over and restart, printing on the screen
7. 04: Gamelogic continue
8. Introduction to AI, with practical examples
9. Introduction to Genetic algorithm
10. Introduction to neural network
11. 05: Initial preparation to gaming AI
12. 06: Importing the AI package and hooking it up – AI learning

Students hands out:

1. Installing Unity
2. Starting a new project in unity, importing materials
3. Controlling the bird – gaming physics. Adding columns
4. Gamelogic – score, game over and restart, printing on the screen
5. Gamelogic continue
6. Initial preparation to gaming AI
7. Importing the AI package and hooking it up – AI learning

What do we learn:

Introduction to Unity – multi platform gaming development environment

Introduction to gaming

Gaming physics

Gaming Control

Based on Unity tutorial: <https://unity3d.com/learn/tutorials/topics/2d-game-creation/project-goals>

Introduction to AI

Introduction to Genetic Algorithm

Introduction to Neural Network

Mixing Genetic Algorithm & Neural Network to teach the machine to play Flappy Bird

Unity Introduction

Initial Setup - 01

1. Create new Unity project, select 2D project.
2. Import the FlappyBird.unitypackage by dragging and dropping into the Assets folder in the project viewer
3. Drag the Scenery Prefab to the Scene window, try to place the object such that the camera frame covers the right area
4. Play the game – the background should scroll and automatically move to create an infinite background effect -> Slide to explain the infinite background idea. No coding at this stage.
5. Drag the Bird prefab into the scene on the left side of the camera frame, in the middle of the Y axis
6. Play the game – the bird will immediately fall to the ground and the ‘Die’ animation will trigger

Instructor notes:

* Unity versions: you might get a note on using a newer version than the one the project was saved with – usually clicking ok will resolve all problems
* SpriteRenderer OrderInLayer property will define which object is in front of which. The sky should be set to 0, columns to 1 and ground to 2. The prefabs should be all set that way, but just in case.

Controlling the Bird: initial coding and a bit of gaming physics - 02

1. Check for input, trigger “Flap” animation, reset velocity and add force:

**if** (Input.GetMouseButtonDown(0)) {

*//...tell the animator about it and then...*

 anim.SetTrigger("Flap");

*//...zero out the birds current y velocity before...*

 rb2d.velocity **=** Vector2.zero;

 rb2d.AddForce(**new** Vector2(0,130));

}

1. When Bird collide, make sure to stop handling input (adding the isDead variable set to false. Set to true in collider check. In FixedUpdate check if bird is dead and if so exit function before reading inputs
2. Duplicate the Ground objects in the Scenery object, move them up outside the camera frame. This will cause the bird to collide and die if going outside the screen.
3. Added a public upForce float variable, use it to set the force on the Y vector. Now you can change the upForce on the Bird object from the editor. Find the Bird script component on the Bird object (should be the last component attached)

Adding the columns:

1. Add a new Script called ColumnSpawn – which will be in charge of spawning new columns
2. Create an empty game object, call it “GameControl” and add the ColumnSpawn script to it.
3. Code the SpawnControl Script, see below. Drag the Column prefab to the component prefab.
4. public **class** ColumnSpawn : MonoBehaviour {
5. public GameObject columnPrefab;*//The column game object.*
6. public **float** spawnRate **=** 3f; *//How quickly columns spawn.*
7. public **float** columnMin **=** **-**1f;*//Minimum y value of the column position.*
8. public **float** columnMax **=** 3.5f; *//Maximum y value of the column position.*
9. private **float** spawnXPosition **=** 10f;
10. private **float** timeSinceLastSpawned;
11. *// Use this for initialization*
12. **void** Start () {
13. timeSinceLastSpawned **=** spawnRate;
14. }
16. *// Update is called once per frame*
17. **void** Update () {
18. timeSinceLastSpawned **+=** Time.deltaTime;
19. **if** (timeSinceLastSpawned **>=** spawnRate)
20. {
21. timeSinceLastSpawned **=** 0f;
22. *//Set a random y position for the column*
23. **float** spawnYPosition **=** Random.Range(columnMin, columnMax);
24. *// Spawn the object and set position.*
25. GameObject column **=** (GameObject)Instantiate(columnPrefab, **new** Vector2(spawnXPosition, spawnYPosition) , Quaternion.identity);
26. }
27. }
28. }

Adding game logic and scoring – 03

Adding Score:

1. We want the score to belong to the Bird instance, since later on we will have multiple birds trying to learn to flap. Add a public score int variable to the Bird script.
2. Add a new script called GameLogic, attach it to the GameControl object. Make a singleton pattern by defining a static member to hold a point to the GameLogic instance. In the Awake() function set this pointer. Add a function called BirdScored(Bird bird) which receives the bird instance that scored (moved between columns). Inside the function add +1 to the bird and Debug.Log the current score.
3. Inside the Column script, inside the OnTriggerEnter2D function, where you already check if the bird collided with the gap between the columns, call the GameLogic BirdScore function.
4. Test your code.

Game Over:

1. In the GameLogic script add a new private variable called gameOver and create a get only property for it called GameOver, set this variable to false. Add a new function called BirdDied that will set the gameOver to true.
2. In the Bird script in the OnCollisionEnter2D function call the GameLogic BirdDied function.
3. In the SCrollingObject script in the update function, check if the GameLogic GameOver property is true and if so set the gameObject rigidBody velocity to zero. This will stop the scrolling of the background and the columns.
4. We should also stop spawning new Columns when the game is over

Removing Columns:

1. Add to Column script a new public variable xPosToDestroy -> this will be set by the ColumnSpawn script to indicate when the column moved out of the camera and can be destroyed.
2. Add an Update function to Column, check if it’s current position.x is less than xPosToDestory and if so call Destroy(gameObject).
3. In the ColumnSpwan script, after instantiating a new Column, set it’s xPosToDestory to -10. (Check if -10 is indeed enough to indicate the column is outside the camera view. A better way would be to add another public variable to the ColumnSpawn and use it to set the value through the editor (more flexible instead of having hard coded values).

Adding Canvas with Score and Game Over

1. Add new UI/Text object. Notice that a new Canvas was added and under it is the new text object. Also, an EventSystem object was added (no need to touch this). In the RectTransform property, click the AnchorPreset icon (left to the position) and then shift+alt click on the left top corner. This will move the text to the left top of the screen. Increase font to 24 and change color to white. Change the Text to “Score: 0”. Make sure the width is at least 160.
2. In the GameLogic script add a new public Text variable called scoreText. This will point to the Text added above. You will need to add **using** UnityEngine.UI statement at the top to use the Text UI object. Remember to assign this in the editor to the newly added Text from the previous step.
3. Now in the GameLogic where we previously added a Debug.Log messege replace with setting the text to the new score.
4. Challenge: Add another Text, center on the screen. On the beginning of the game it should be empty, when the game is over it should say Game Over

Completing the game: Implementing a start screen. Implementing a restart after gameover. 04

1. Renamr the GameoverText to CenterText, be used for both the start and end game text.
2. When the game starts, create a title and asked the player to press space to start
3. To pause the game on start we will use Time.timeScale = 0 in the start method. Lets also add a bool variable called isPaused and set it to true on start. (better than checking the timescale value and we might want to separate those two actions)
4. In the GameLogic update function, check if the game is paused and if so check if the user press ‘return’. If he did: remove the welcome text (set it to “”), set timescale back to 1 and set isPaused to false.
5. This works in the start of the game, now we want to restart after the bird died: In the GameLogic BirdDied function do the following: set timescale to 0, set isPaused to true, and set the CenterText to game over messege.
6. Now we are missing a restart of the bird and columns:
7. Bird restart: Create a public Restart function in the Bird script which does the following: set its position to its initial position (you can save the initial position in the awake function. Set isDead to false, set score to zero, reset the rigidbody velocity and angular velocity (in case the collision caused it some rotation movement), reset its rotation back to Quaternion.identiy. Here is the code:

public **void** Restart() {

        transform.position **=** initialPosition;

        isDead **=** false;

        score **=** 0;

        rb2d.velocity **=** Vector2.zero;

        transform.rotation **=** Quaternion.identity;

        rb2d.angularVelocity **=** 0;

    }

1. Provide access to the Bird to the GameLogic script and call the Bird Restart when the user presses ‘return’ if the game was in game over state.
2. For the columns to be reset we can do the following. First, in the update function when we create a new column we will also set its parent to the ColumnSpawn transform (the gameObject that the script attaches to). This will allow us to find and delete the column on restart. Now add a restart function as follows:

public **void** Restart() {

        timeSinceLastSpawned **=** spawnRate;

        List<Transform> childList **=** **new** List<Transform>();

**for** (**int** i **=** 0; i **<** transform.childCount; i**++**) {

            childList.Add(transform.GetChild(i));

        }

**foreach** (**var** trans **in** childList)

        {

            Destroy(trans.gameObject);

        }

    }

1. Notice that we first create a list to hold the column and only then call Destroy. This is a safe practice when deleting objects in an array that might change the array we are traversing.
2. Call this Restart() from the GameLogic script and the user press ‘return’ if the game is in game over state.
3. For reference, here is the GameLogic code so far. Notice that instead of setting timescale to zero immediately when the Bird dies, we call Invoke with 1 second delay- this allows the dying simulation to take place.

public Bird bird;

    private const **string** WellcomeText **=** "Flappy Bird, press Return to start.";

    private const **string** GameOverText **=** "Game Over, press Return to start.";

*// Use this for initialization*

**void** Awake() {

        instance **=** **this**;

    }

**void** Start () {

        centerText.text **=** WellcomeText;

        Time.timeScale **=** 0;

        gamePaused **=** true;

    }

*// Update is called once per frame*

**void** Update () {

**if** (gamePaused **==** true) {

**if** (Input.GetKeyDown(KeyCode.Return)) {

*// Un pause the game either start (first time playing), or restart*

**if** (gameOver **==** true) {

*// Restart game:*

*// 1. restart Bird:*

                    bird.Restart();

                    GetComponent<ColumnSpawn>().Restart();

                }

                centerText.text **=** "";

                Time.timeScale **=** 1;

                gamePaused **=** false;

            }

        }

    }

    public **void** BirdScored(Bird bird) {

        bird.score **+=** 1;

*//Debug.Log("Score: " + bird.score);*

        scoreText.text **=** "Score: " **+** bird.score;

    }

    public **void** BirdDied() {

*// Let provide an addition 1 second for the simulation to run to show the bird*

*// fall after hit and only than pause the simulation*

        gameOver **=** true;

        centerText.text **=** GameOverText;

        gamePaused **=** true;

        Invoke("PauseTime", 1);

    }

**void** PauseTime() {

**if** (gamePaused**==**true)

            Time.timeScale **=** 0;

    }

Introduction to AI

Usages of AI in the world, not only games

Maybe after the random AI, add another step before genetic algorithm

Introduction to genetic algorithms

Introduction to neural networks

Initial preparations to hook the game into basic AI and create multiple Bird instances: 05

1. Let’s create a new Script called AILogic. This script doesn’t need to inherit from MonBehaviour. Create a public function called FlapWings that received two float parameters and return bool. This function will decide based on those two parameters if to “Flap the bird Wings”, meaning simulate the user pressing space which will add force on the Y direction.
2. For now let’s assume that the function receives the x/y position of the bird on those two parameters and we return true based on the y position of the bird and a random value. See the initial implementation below:

public **class** AILogic {

    public **bool** FlapWings(**float** param1, **float** param2) {

**if** (param2 **<** 0 **&&** UnityEngine.Random.Range(0,100)**>**90f) {

**return** true;

        }

**if** (param2 **>=** 0 **&&** UnityEngine.Random.Range(0,100)**>**97f) {

**return** true;

        }

**return** false;

    }

}

1. Inside the Bird script add the following variable: AILogic ai **=** **new** AILogic(). Inside the update function, where we check for user input, replace that code with a call to ai.FlapWings() passing the bird x/y position. Run the game and see how the AI performs. To make the bird initial state more random and colorful, change its awake function as follows:

**void** Awake() {

*//Get reference to the Animator component attached to this GameObject.*

        anim **=** GetComponent<Animator> ();

*//Get and store a reference to the Rigidbody2D attached to this GameObject.*

        rb2d **=** GetComponent<Rigidbody2D>();

**float** xPosition **=** UnityEngine.Random.Range(**-**8f, **-**6.5f);

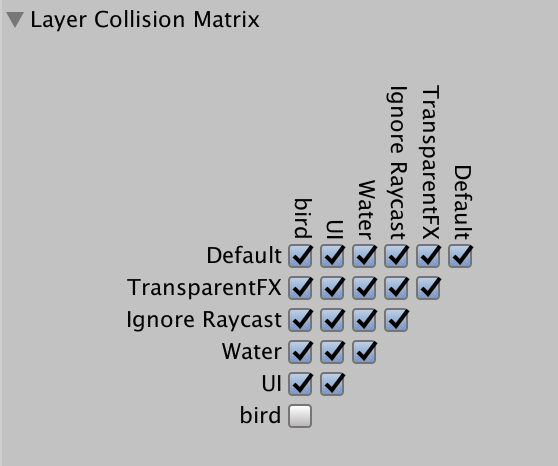
**float** yPosition **=** UnityEngine.Random.Range(**-**2f, 2f);

        initialPosition **=** **new** Vector2(xPosition, yPosition);

        gameObject.GetComponent<SpriteRenderer>().color **=** Random.ColorHSV();

        transform.position **=** initialPosition;

    }

1. Before we continue to create multiple birds and let the AI race each other, we need to make sure the bird will not collide with each other. To do that we need to add a new layer and assign the bird prefab to it. Select the Bird prefab and on the properties window on the right top side under Layer select Add Layer… select one of the empty entries and create a new layer called bird. Now make sure the rest of the objects are not set to this layer by mistake. Double check the rest of the prefabs and the scenery objects that are already in the scene – they should all be under default. Next open the physics2d properties (under the Edit/Project Setting menu). Then remove the option for the bird to collide with itself, see the below picture: 
2. Now lets change the GameLogic script to instantiate multiple birds: Delete the current Bird object in the scene (we are going to create the birds from the script). Inside the GameLogic script add a list variable to hold the birds: List<Bird> birds **=** **new** List<Bird>() (Remove the previous Bird variable). Also add a GameObject variable to point to the bird prefab and an int variable to specify how many birds we want. In the editor drag the bird prefab to the GameLogic new variable.
3. Create an Instantiate function and call it from the GameLogic Start function, here is how to instantiate the birds:

**void** InstantiateBirds() {

**for** (**int** i **=** 0; i **<** numberOfBirds; i**++**) {

            birds.Add(Instantiate(birdPrefab, Vector2.zero, Quaternion.identity).GetComponent<Bird>());

        }

    }

1. We use Unity Instantiate function the same way we did with the columns. We also get a pointer to the Bird script component and add it to the birds list.
2. We need to fix the code that calls the bird.Restart() function when the game restarts. Instead of calling a single bird, create a foreach statement and restart all the bird on the list.
3. Run the game to view the results. Notice that we are still missing few critical fixes: 1) score should hold the best score. 2) the game should end when all the birds dies. Lets fix that.
4. To fix that we need 2 new variable, one to hold maxScore and one to hold alive birds. On the BirdScore function check if current bird score is bigger than max and update max. On the BirdDied function reduce the number of alive birds and only when reaching zero end the game. Remember to initialize and reset these variable in the correct places.

Hooking up Genetic algorithm with Neural Network. 06

1. Drag the ai.unitypackage into the product, this will bring 3 new script: Mutation script is just an interface of what a mutation needs to implement, Genetic is an implementation of the genetic algorithm and NeuralNetwork is an implementation of a neural network that uses the Mutation interface such that the Genetic algorithm can have mutations that are each an instance of a neural network.
2. Changes to the Bird script:
   1. Add public member for: 1) int mutationIndex -> will be used to set the fitness 2) make sure isDead Boolean is public so that it can be accessed from the GameLogic
   2. Add a public Boolean called flapOnNextFrame – use this to test if to flap the bird wings in the update function. The AI will set this variable in the GameLogic. Make sure to initialize this variable to false and to set it back to false after flapping the wings
   3. Make sure to exit the update and collision functions immediately is the isDead variable is set to true
3. Changes to the GameLogic:
   1. Lets add the support for the Genetic algorithm. Add the following public variables: 1) **int** numOfUnitsInGeneration **=** 40 – will be used to determine how many units we have in the population 2) **float** topUnitsPercentage **=** 0.25f – determines the percentage of units we don’t change from one generation to another 3) **float** randomUnitsPercentage **=** 0.10f – the percentage of units we will completely randomize each generation.
   2. Add a private variable to track the generation number
   3. Add a private float variable to track the distance pass -> update this variable in the update function by adding Time.timeDelta; (so this track time more than distance)
   4. Add an instance of the Genetic class, call it genetic;
   5. Add a method as follows: This method will be used to initialize the Genetic class and tells it how to instantiate a new mutation

Genetic genetic;

    Mutation CreateNetwork(Mutation parent**=**null) {

**return** **new** NeuralNetwork(2, 1, **new** **int**[1]{6});

    }

* 1. Change the InstantiateBirds() function to assign the mutationIndex to each bird, this is simply the index of the for loop
  2. When the bird dies, add the following line: genetic.SetFitness(bird.mutationIndex, distance) – this will set the mutation fitness.
  3. Repurpose the centerText to show the generation number of the screen instead of the game start and end texts.
  4. Remove all references to the pause of the game – we don’t need to pause it anymore.
  5. The biggest change in GameLogic is the update function, here is the complete function:

**void** Update () {

**if** (**!**gameOver) {

*// \*\*\* AI Control*

            distance **+=** Time.deltaTime;

*//Debug.Log(distance);*

*//distanceText.text = "Distance: " + distance.ToString("F1");*

**for** (**int** i **=** 0; i **<** numOfUnitsInGeneration; i**++**) {

**if** (birds[i].isDead **==** false) {

**float**[] inputs **=** **new** **float**[2];

                    columnSpawn.GetNearestColumn(birds[i].transform, out inputs[0], out inputs[1]);

*//float[] inputs = new float[2] { columnPool.GetHorizontalDistance(bird[i].transform) / 5f , columnPool.GetHeightDifference(bird[i].transform) / 2f };*

**float**[] result **=** genetic.ActivateBrain(i, inputs);

*//Debug.Log("Inputs: " + inputs[0] + "," + inputs[1] + " outputs: " + result[0]);*

**if** (result[0] **>** 0.5f) {

                        birds[i].flapOnNextFrame **=** true;

                    }

                }

            }

        } **else** {

*// \*\*\* AI Control*

            genetic.EvolvePopulation();

            generationNum **+=** 1;

            generationText.text **=** "Generation: " **+** generationNum;

*//generationText.text = "Generation: " + generationNum.ToString();*

            Restart();

        }

    }

1. In ColumnSpawn add the following code: this code calculates the distance to the nearest column and it’s height difference from the birds location. We use this function in the GameLogic update to as the inputs to the ActivateBrain() function

**float** deltaX **=** 0.80f;

    public **int** GetNearestColumn(Transform from, out **float** horizontalDistance, out **float** heightDifference) {

**int** bestFound **=** **-**1;

        horizontalDistance **=** 30;

        heightDifference **=** from.position.y **-** 0.558f;

**for** (**int** i **=** 0; i **<** transform.childCount; i**++**) {

**float** testDistance **=** (transform.GetChild(i).transform.position.x**+**deltaX) **-** from.position.x;

**if** (testDistance **>**0 **&&** testDistance**<**horizontalDistance) {

                bestFound **=** i;

                horizontalDistance **=** testDistance;

                heightDifference **=** from.position.y **-** transform.GetChild(i).transform.position.y;

            }

        }

**return** bestFound;

    }

1. Challenge: Add a new variable to the GameLogic to end a generation when score reaches a certain value, hint: make sure to call BirdDied on each of the still alive birds.