DTT
APPS WEB GAMES

SINCE 2010

# Anagram word Finder

Documentation | 18-05-2022

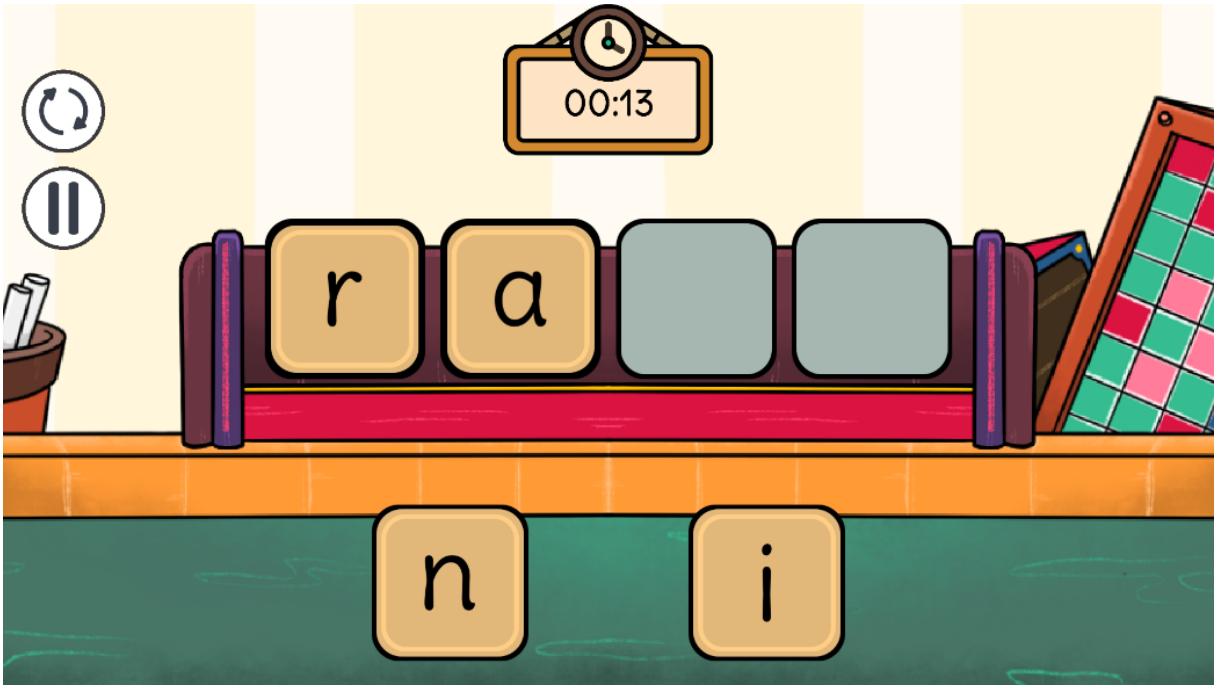
# Table of Contents

# 1. Get started quickly

- Create a gameobject in your scene and add the AnagramManager component to it. Then, add the AnagramUI component as well and drag the component into the 'Anagram UI' field on the AnagramManager.
- Create a prefab and add the AnagramLetter and **RectTransform** component to it. Add a **Text** component to the prefab as well and drag it into the 'Text' field on the AnagramLetter component. Drag this prefab into the 'Letter Prefab' field on the AnagramUI component.
- Create another prefab and add the AnagramSnapPoint and **RectTransform** component to the prefab. Drag this prefab into the 'Snap Point Prefab' field on the AnagramUI component.
- Now create a canvas in your scene. No specific settings are required for the canvas. Add a gameobject to the canvas and add the **Horizontal Layout Group** component to it. Drag this object into the 'Snap Layout Group' field on the AnagramUI component.
- Add another gameobject to the canvas, and add the SnapBoard component to it. Drag this object into the 'Letter Snap Board' field on the AnagramUI component.
- Create 2 more gameobjects in the canvas. Drag one of them into the 'Drag Holder' field of the AnagramUI, and the other into the 'Drop Holder' field of the AnagramUI.
- Right click in the project tab of Unity, and right click. Go to 'Create/DTT/MiniGame/Anagram/Config'. Inspect the newly created AnagramConfig object and press 'Regenerate Anagrams'. Now drag the object into the Config field of the AnagramManager component.
- You can now start the game by either setting the Start On Awake field on the AnagramManager to true, or calling AnagramManager.StartGame.

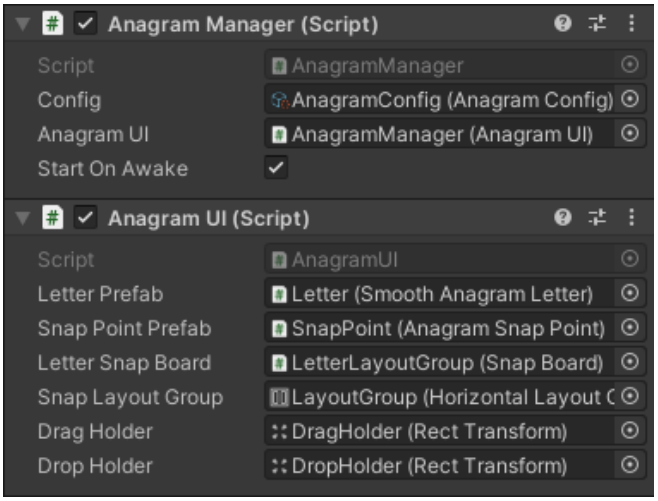
# 2. Introduction

DTT Anagram word Finder is an Unity asset that allows you to easily implement an anagram word finder game into your project. The asset allows you to customize the playing field and difficulty of the game. Using a text asset you can define your own words that will be used in the game. The asset also has a default text file containing almost a 1000 different words to get you started quickly. To create levels, you can use Scriptable Objects that allow you to change the difficulty and which words will be used for this level. All this allows for an easy and fast way to implement a simple anagram word finder game into your project.



# 3. Set-Up

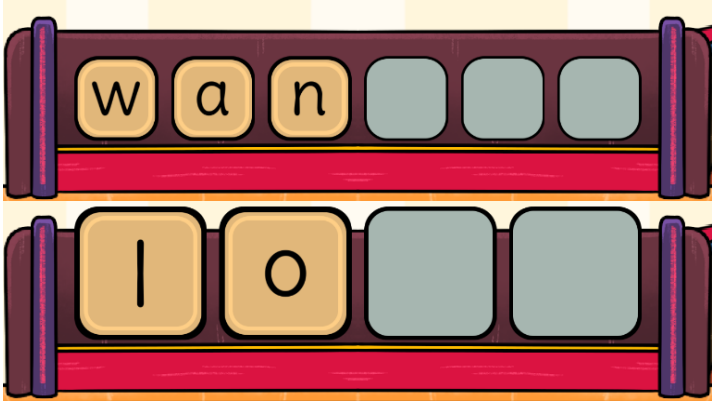
An example on how to implement the game can be found in the demo scene under the 'Demo/Scenes' folder.




1. First add a gameobject to your scene. Add the AnagramManager component to the gameobject. This component will handle the core game loop of the game.
2. Add another gameobject and add the AnagramUI component to it. This component will handle the creation and behavior of the UI elements of the game. Drag this component into the 'Anagram UI' field on the AnagramManager component.
3. Create a prefab and add the AnagramLetter component to it. Optionally, you can add the SmoothAnagramLetter component instead. The smooth component implements a basic smooth snap animation to the letter. After adding either one of the components, add a **Text** component to the prefab and drag it into the 'Text' field of the AnagramLetter component. Then drag the prefab into the 'Letter Prefab' field on the AnagramUI component.
4. Create another prefab and add the AnagramSnapPoint component to it. This component handles holding the AnagramLetter prefab. Drag the prefab into the 'Snap Point Prefab' field on the AnagramUI component.
5. Now create a canvas in your scene. The canvas has no required settings.
6. Add a gameobject to the canvas and add the **Horizontal Layout Group** component to it. This layout group will hold all the points the letters of the game can snap to. Drag this object into the 'Snap Layout Group' field on the AnagramUI component.

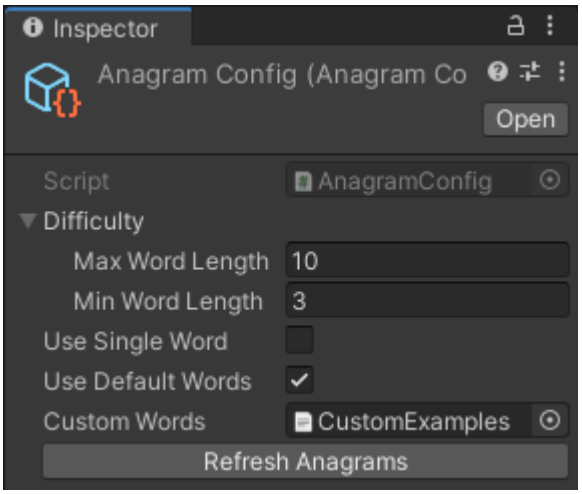
   NOTE: The Control Child Size field on the Layout Group will be set to true by the AnagramUI. The letter objects will also have their width and height set to the

same width and height of the created snap points on the layout group. This is done to keep the scaling of the snap points and letters consistent when there are longer words. We recommend using the **Aspect Ratio Fitter** component with the 'Aspect Mode' set to 'Width Controls Height' to keep the UI width and height ratio the same.




7. Add another gameobject to the canvas and add the SnapBoard component to it. This component will handle holding the letters when they are not snapped to the 'Snap Layout Group'. The letters will be evenly spread across the width of this object. Drag the SnapBoard object into the Letter Snap Board field on the AnagramUI component.
8. Now add 2 more game objects to the canvas. Drag one into the 'Drag Holder' field on the AnagramUI component. This object will hold the letters that are currently being dragged. Drag the other object into the 'Drop Holder' field on the AnagramUI component. This object will hold the letters that are currently dropped on the letter board.

9. Now that the UI is set up, you can define the words and difficulty for your anagram word finder game. To do this, go to the project tab of unity. Right click, and go to 'Create/DTT/MiniGame/Anagram/Config'. This will create an **AnagramConfig** scriptable object. On this object, you can define the minimum and maximum length of the words in the game. Drag this object into the 'Config' field on the AnagramManager component.



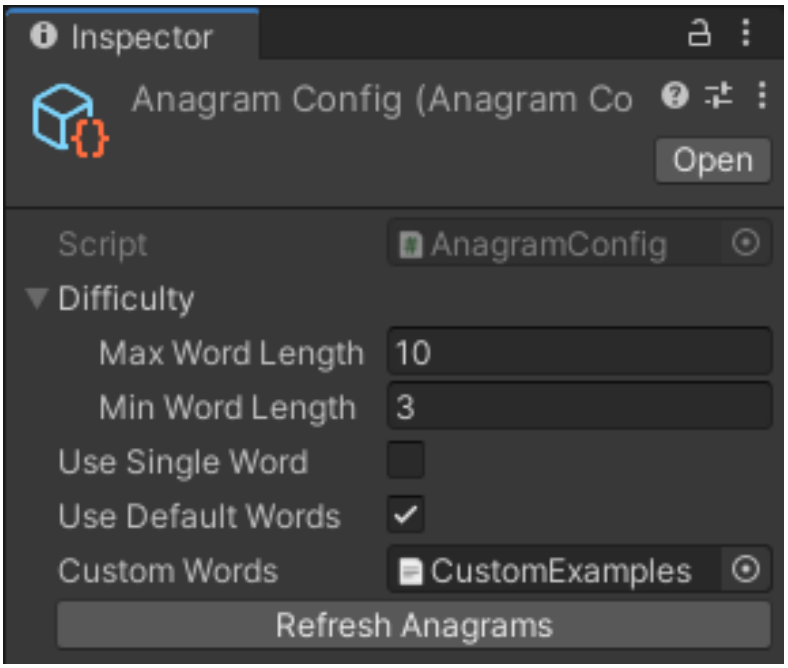

   You can also define which words to use in the game. If you set the 'Use Default Words' field to true, the words from the DefaultWords text asset will be used. This file can be found under the Runtime folder. You can also create your own text asset to define your own words. Words need to be separated by either a comma, or set on a new line. You can use the 'CustomExample.txt' file in the Demo folder as an example. The custom text asset can be dragged into the 'Custom Words' field.

   Now press the 'Refresh Anagrams' button. This will sort all the words in the default and/or custom text asset into collections of words that are anagrams of each other. When the game starts, a random collection of anagrams will be chosen and used for the game.

   NOTE: When the user completes a word, it will check if the word is equal to any of the anagrams in the collection. For example, if the words 'ear' and 'are' are in the text assets, either of these words will be seen as correct when playing the game. If you only want 1 word to be correct, set the 'Use Single Word' field to true on the AnagramConfig object.

10. To finally start the game, either set the Start On Awake field on the AnagramManager to true, or call AnagramManager.StartGame.

# 4. Editor




The **AnagramConfig** inspector window allows you to set the difficulty and generate anagrams for the anagram word finder game.

1. *Max Word Length*
   The maximum amount of characters allowed in a word.
2. *Min Word Length*
   The minimum amount of characters allowed in a word.
3. *Use Single Word*
   Only allows a single word as a correct answer. Anagrams of this word won't be seen as correct.
4. *Use Default Words*
   Whether the words from the DefaultWords.csv file should be used. This file can be found under the Runtime folder.

5. *Custom Words*

   Text asset containing your own words. Can be left empty if the default words are used. Words in the asset must be separated with either a comma or set on a new line.

```
1   hello world, big person, custom
2   straight, forward, usage
3   rear, rare
```

6. *Refresh Anagrams*

   This button generates a list of collections containing anagrams found in the default words and/or custom words text assets. Make sure to press this button when you make changes to the custom text asset, or when you disabled the default words.

# 5. API

## AnagramManager

```
// Reference to the current Anagram Manager in the scene.
private AnagramManager manager;

// Reference to your custom Anagram Config.
private AnagramConfig myConfig;

private void Start()
{
    // Starts the game.
    manager.StartGame();

    // Starts the game with the given Anagram Config.
    manager.StartGame(myConfig);

    // Pauses the game and disables interaction.
    manager.Pause();

    // Continues the game from the paused state.
    manager.Continue();

    // Forces the game to end.
    manager.ForceFinish();

    // Gets the possible anagrams of the currently active game.
    AnagramCollection currentAnagrams = manager.CurrentAnagrams;

    // Whether the game is currently playing.
    bool isGameActive = manager.IsGameActive;

    // Whether the game is currently paused.
    bool isPaused = manager.IsPaused;

    // The duration of the current active game.
    float timePlayed = manager.TimeElapsed;
}
```

```
private void OnEnable()
{
    // This event is invoked when the game starts.
    manager.Started += MyStartMethod;

    // This event is invoked when all letters are snapped to the snap points.
    // The event return a result which contains the word that was filled in,
    // and whether it was a correct word.
    manager.Finish += MyFinishedMethod;
}

private void MyStartMethod()
{
    // My start behaviour...
}

private void MyFinishedMethod(AnagramResult result)
{
    // The CorrectWord bool is true when the user found a correct word.
    bool correctWordWasFound = result.CorrectWord;

    // The Word string is the word that was found by the user.
    string wordFound = result.Word;
}
```

## AnagramSnapPoint

```
// Reference to a snap point.
private AnagramSnapPoint snapPoint;

// Reference to a letter.
private AnagramLetter letter;

private void Start()
{
    // Snaps the given letter to the snap point.
    snapPoint.SnapLetter(letter);

    // Snaps the letter to a given world position.
    letter.SnapToPosition(Vector2.one);
}

private void OnEnable()
{
    // This event is invoked when a letter is dragged on the snap point.
    snapPoint.LetterSnapped += MyLetterSnappedBehaviour;

    // This event is invoked when a letter is replaced with another letter.
    snapPoint.RemovedLetter += MyLetterRemovedBehaviour;
}

private void MyLetterSnappedBehaviour()
{
    // Gets the current letter that is snapped to this point.
    AnagramLetter currentLetter = snapPoint.CurrentLetter;
}

private void MyLetterRemovedBehaviour(AnagramLetter removedLetter)
{
    // Do stuff...
}
```

## AnagramLetter

```
// Reference to a letter.
private AnagramLetter letter;

private void Start()
{
    // This event is invoked when the letter is picked up.
    letter.PickUp += MyPickUpBehaviour;

    // This event is invoked when the letter is dropped.
    letter.PickUp += MyDropBehaviour;

    // The letter this component represents.
    char character = letter.Letter;
}

private void MyPickUpBehaviour(AnagramLetter letter)
{
    // Do stuff...
}

private void MyDropBehaviour(AnagramLetter letter)
{
    // Do stuff...
}
```

You can also make your own implementation of the **AnagramLetter** to override the snap animation. An example of this can be found in the **SmoothAnagramLetter** script under 'Runtime/UI'.

## AnagramConfig

```
// Reference to the config.
private AnagramConfig config;

private void Start()
{
    // Allows you to programatically change the difficulty of the game.
    config.SetDifficulty(new AnagramDifficulty
    {
        minWordLength = 2,
        maxWordLength = 4
    });
}
```

# 6. Known Limitations

- **Snap Board:** The **SnapBoard** component currently has no customization, and only allows one layout appearance for the letter.

# 7. Support and feedback

If you have any questions regarding the use of this asset, we are happy to help you out. Always feel free to contact us at:

unity-support@d-tt.nl

*(We typically respond within 1-2 business days)*

We are actively developing this asset, with many future updates and extensions already planned. We are eager to include feedback from our users in future updates, be they 'quality of life' improvements, new features, bug fixes or anything else that can help you improve your experience with this asset. You can reach us at the email above.

Reviews and ratings are very much appreciated as they help us raise awareness and to improve our assets.

### DTT stands for Doing Things Together

DTT is an app, web and game development agency based in the centre of Amsterdam. Established in 2010, DTT has over a decade of experience in mobile, game, and web based technology.

Our game department primarily works in Unity where we put significant emphasis on the development of internal packages, allowing us to efficiently reuse code between projects. To support the Unity community, we are publishing a selection of our internal packages on the Asset Store, including this one.

More information about DTT (including our clients, projects and vacancies) can be found here:

https://www.d-tt.nl/en/