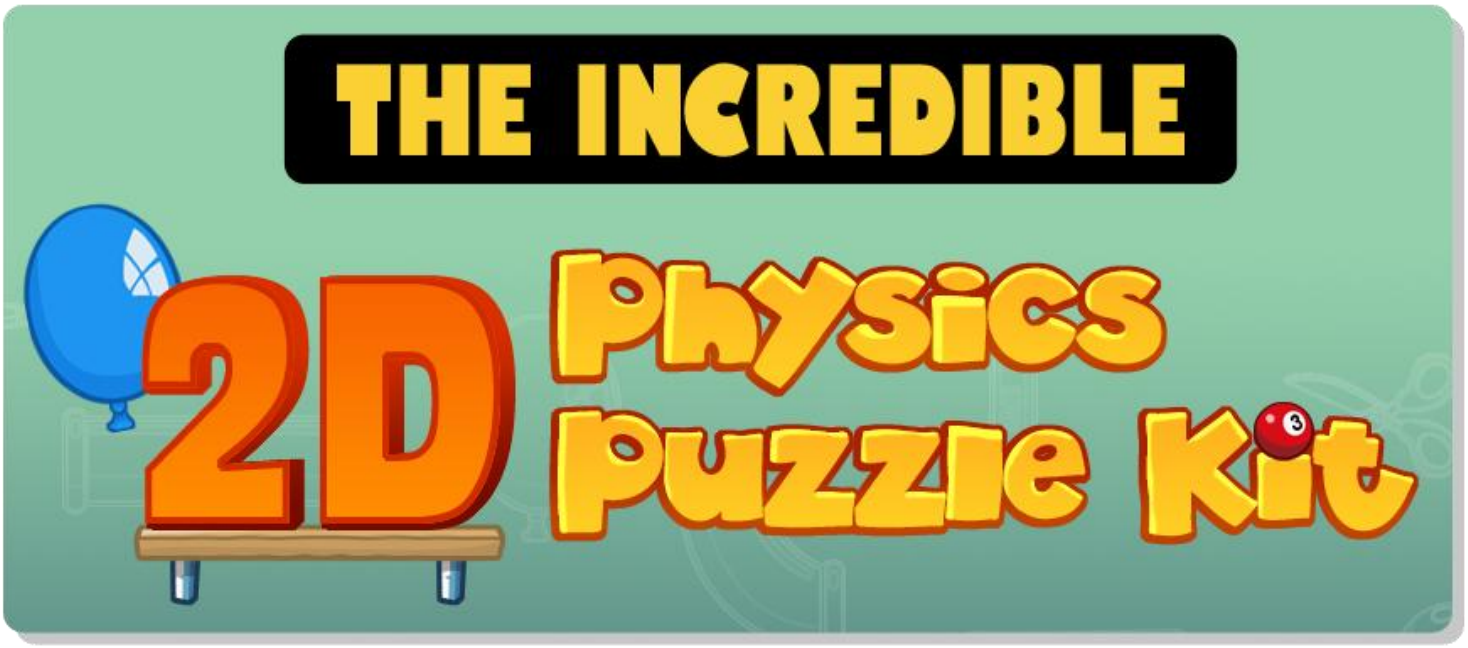


Thank you for purchasing **2D PHYSICS PUZZLE KIT**.

If you have any problems, suggestions, feature requests or questions just let us know and we will try to answer them as soon as possible!

Get in touch via our forum thread or email: dreeka@hisemailaccount.com

CONTENTS

2D PHYSICS PUZZLE KIT

Note: Click on the links to jump to the specified sections in the documentation

1. Overview

Thank you for purchasing the 2D Physics Puzzle Kit!
The goal of this documentation is to showcase how the main features work in the kit.
For more information, you can check the detailed comments in the code, or contact me. If you have any feedback or questions just let us know and we will try to answer them as soon as possible!

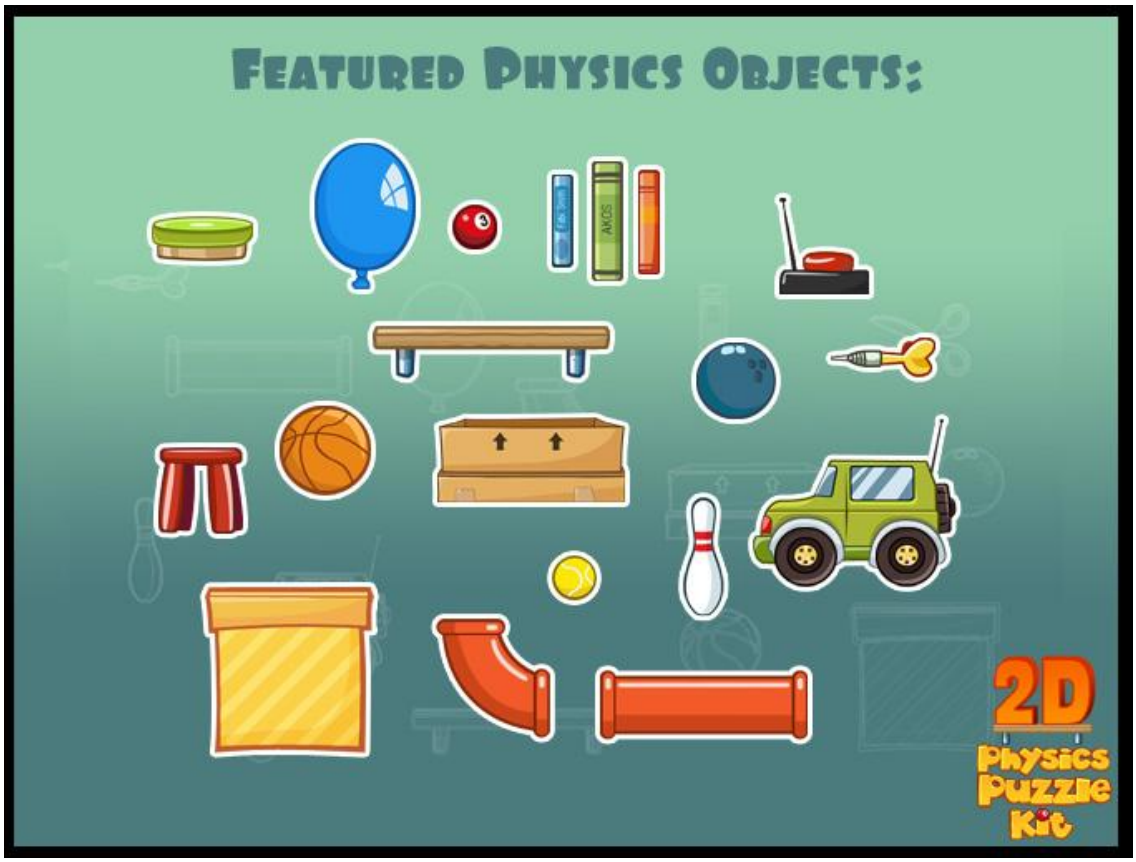
E-Mail: mako752@gmail.com

2. Gameplay

Each level starts with game items already placed in the map. Among these items is the target item. The goal of the game is to guide the target item to a predetermined location. To guide the target, the player can put new items to the level from the toolbox, while in planning mode. Each level has 3 stars, which the player can collect. To collect a star, an item has to collide with it.

Each level has 2 game modes, *planning* and *playing*. The game starts in planning mode. In this mode, the items are not moving and the player can place items from the toolbox or modify already placed ones. Playing mode is the opposite. The items are moving and the player can't place or modify items. Players can switch between planning and playing mode as they wish.

The kit contains the following items:



3. PROJECT SETUP

- Create a new 2D project, and download or import the kit.
- Go to **Edit** -> **Project Settings** -> **Physics 2D**
- In the layer collision matrix, uncheck Mask -> GameObject
- In the layer collision matrix, uncheck GameObject -> GUI

4. CLASS OVERVIEW

Manager classes (singleton implementation):

- **Main Menu Manager**: Handles the main menu GUI.
- **Input Manager**: Captures and manages the player input.
- **Level Manager**: Manages the objects in the scene, the level states and the saving.
- **GUI Manager**: Manages the graphics user interface.
- **Toolbox Manager**: Handles toolbox interaction and toolbox data.
- **Goal Manager**: Checks for goal completion.
- **Feedback Manager**: Manages the graphical feedback for object movement/rotation.

Game item classes (in most cases, the items hold both Object Base and Physics Object):

- **Object Base**: Base class, each item has one. Manages the position, rotation, player movement and rotation of the item. Also manages switching between game modes.
- **Physics Object**: Base class for items which are affected by gravity.
- **Small Ball**: Inherited class from Physics Object. The small balls use it.
- **Balloon**: Inherited class from Physics Object. The balloon uses it.
- **RC Car**: Inherited class from Physics Object. The RC Car uses it.
- **RC Controller**: Inherited class from Physics Object. The RC Controller uses it.
- **Star**: Inherited class from Object Base. The Star uses it.

Other Classes:

- **Toolbox Item Type**: The toolbox version of the game items uses it.
- **Level Select Button**: The level buttons use it in the main menu.
- **Set Sorting Layer**: Modifies the sorting layer and order of the item.
- **Trigger Registerer**: Sends trigger events to parent.

5. GAME ITEM LOGIC

The game items are in the GameObject layer, and use the GameObject tag. Each game item contains the following components: **Transform**, **Sprite Renderer**, **Rigidbody2D**, **Collider2D**, and **Object Base**. Each item has a toolbox version, which will be discussed in the Toolbox part.

Static items (shelf, pipes, dart arrow)

These items do not move, and they are fixed to their position in playing mode. Only the player can move them in planning mode, if they are picked from the toolbox.

Basic Physics Objects (books, crate, and so on)

These items have gravity, so they can move on their own, or other items can push them. They have the **Physics Object** component as an addition to the base components.

Big Balls (basketball, bowling ball)

They have every property the basic physics objects have, with an extra. Their colliders have a physics material, which gives the bounciness and friction.

Small Balls (tennis ball, billard ball):

They have every property the big balls have, with an extra. They can travel in the pipes. When in a pipe, their bounciness and friction is disabled, so they can easily travel inside the pipes.

Balloon:

The balloon keeps accelerating upward as long as it can. A maximum rotation and maximum rotation force is defined for it. When the rotation of the balloon is not zero, a rotation force is applied to the balloon, in the opposite direction of the current rotation. This keeps the balloon from rotating around its axis.

If the balloon collides with something, after the collision is over, the horizontal speed of the balloon is removed in a small time. This makes it to easily navigate the balloon with obstacles.

If the balloon collides with a dart arrow, it explodes.

RC Controller:

This object controls the RC Car. The speed of the car depends on how much the controller's knob is pushed down. At the start of the level, the starting distance between the knob and the body of the controller is calculated. It is also defined that how much the controller can go down. If an object is in the knob, it will push it down. In this case, the knob applies a small upward force (small enough to not push the item away, but move the knob up once the item is not pushing it down).

The controller sends a signal to the RC Car. The strength of the signal is determined by how much the knob is pushed down. If the signal is strong enough, the *signal out particle* shows the movement of the signal.

RC Car:

The speed of the car is depending of the strength of the incoming signal. If the signal is strong enough, the *signal in particle* shows the movement of the signal.
The body and the wheels of the car are separate physics objects. They are joined together by a Hinge Joint 2D component. This way, the wheels of the car can rotate on their own, and the wheels and body still moves together.

Star:

The star's collider is a trigger collider. If an object enters this zone, while the level is in playing mode, the pickup particle plays, and the star is picked up.

6. SWITCHING BETWEEN GAME MODES

Switching between game modes can be done by pressing the button on the top right corner of the screen. The Input Manager detects the input and forwards it to the GUI Manager. The GUI Manager shows/hides the UI elements and notifies the Level Manager. Then the Level Manager loops through the items in the screen, and calls their Enable or Reset function, based on the current game mode.

7. INPUT

Every input in the levels are registered and handled by the Input Manager. It can be in the following states: waitingForInput, scrolling, moving, and rotating.

On every frame, it detects what is under the mouse by raycasting. Then a function is called, based on the state of the manager.

If there is no active input, then the state is waitingForInput. In this state, the Input Manager is waiting for a proper input from the player. If the player is clicked on a button, the Input Manager notifies the GUI Manager.

If the player clicked on the toolbox background, or on an item in the toolbox, then the Input Manager prepares to scroll the toolbox. The exact goal of the player is unknown at this point.

There are 2 possibilities: the player wants to scroll the toolbox, or wish to remove an element from the toolbox. To determinate the goal, the Input Manager uses the following method: if the player moves the mouse up, out from the toolbox, and the mouse distance is not too far from the starting input, then the player wishes to remove the selected item from

the toolbox. Or if the player moves the mouse too far away from the starting point of the input, then the player want to scroll the toolbar.

If the player is clicked on an item in the level, and the selected item is placed by the player, then the item can be moved. While moving, the distance between the centre of the item and the starting input position will always remain the same.

If the player is clicked on the feedback, and the item can be rotated, then the manager prepares to rotate the selected item. The manager stores a starting vector, which starts from the centre of the item, and points at the starting input position. Another vector is calculated, which starts from the centre of the item, and points at the actual input position. Then the manager calculates the angle difference between the two vectors, and rotates the item based on this angle.

8. GUI

The GUI is managed by the GUI Manager. The other managers call its public functions to show/hide parts of the GUI, based on the current events. Moving the elements is done by using the MoveMenuElementBy function.

9. TOOLBOX

Every game item has a smaller toolbox version. These toolbox versions use the Toolbox Item Type class. Their only goal is to be used in the toolbox, and they do not have the behavior of their normal counterpart.

At the start of the level, these toolbox items are creating a predetermined number of clone items from their normal counterpart and they move these clones under them in the hierarchy, and disable them. Then the Level Manager gives the toolbox versions to the Toolbox Manager.

The Toolbox Manager positions these elements on the toolbox strip, with equal space between then.

Because the toolbox can be scrolled, those items that leave the toolbox strip either to the left, or to the right, have to be hidden. This is done with a masking technique. On the left and right side of the toolbox, an invisible mask is placed, which uses a special shader. These items hide everything under them. This way, once an item leaves the toolbox, it goes under the mask, and gets partially or fully invisible.

But because everything is hidden under the masks, part of the background is hidden as well. To fix this, the game uses a dual camera setup. There are two cameras. One camera renders the background, and the level items, the other camera renders the GUI and the masks.

If the user removes an item from the toolbox by clicking on the toolbox version, then the selected toolbox version gives one of its child items to the player (these items were created at the start of the level, and these items are the fully functional game items).

If a toolbox item doesn't have any more childs, it removes itself from the toolbox. This can be done in two ways. If there are no other items in the toolbox, then this item is disabled and the toolbox closes. If there are more items, then the item is disabled and the other items to its left move to the right to their now position (to keep the equal distance between items). If the item can fit into the toolbox without scrolling, then the toolbox shrinks itself to match the new width of the items.

If an item returns to the toolbox, then the Toolbox Manager checks for its toolbox version. If the toolbox manager finds it in the strip, then the items are simply given to its toolbox variant. If it is disabled, then the toolbox manager reactivates it and places it back to the toolbox. If this is the only element, then its position is calculated from the toolbox button. Otherwise, it is calculated from the most right element. If the item can fit into the toolbox without scrolling, then the toolbox enlarges itself to match the new width of the items.

When the level is reset, the Toolbox Manager loops through the active and inactive toolbox items and resets them. Then the strip and the containers are emptied. The Level Manager passes the toolbox items again, in their original order, and the toolbox is rebuilt based on that.

10. GOAL

To check the goal completion, the Goal Manager receives data from every collision, OnTriggerEnter, and balloon explosion. Then the goal manager checks, that the target object is interacted with the goal position or not. If yes, the goal is completed and the Level Manager is notified.

11. SAVING

Saving is done by the built in PlayerPrefs system. It allows the storage of data based on keys. The level data is stored in the following format: the key is the level number, and its value is the level state. -1 means that the level is not completed. 0, 1, 2 or 3 means that the level is completed with 0, 1, 2 or 3 stars.

12. CREATING NEW LEVELS

If you would like to create new levels, open the empty level scene, which can be found next to the pre created levels.

You can put the game items you wish to appear on the scene under LevelManager -> LevelElements. These items are fixed, and can't be modified by the player. You can assign toolbox items to the toolbox as well. You have to put the toolbox items under LevelManager -> ToolboxElements.

You can find the normal and toolbox items in the 2D Physics Puzzle Kit -> Prefabs folder.

13. CREATING NEW ITEMS

To create new game items, following these steps:

- Import the new texture to the Textures -> Objects -> Normal folder as a sprite.
- Drag the texture to the scene to create an object.
- Set the sorting layer, tag, and layer to GameObject.
- Set a custom sorting order.
- Assign a 2D rigidbody and collider(s) to the object
- Assign the ObjectBase script to the object (or inherit a new script from it, and assign that).
- Assign the PhysicsBase script to the object, if needed (or inherit a new script from it, and assign that).
- Save the item as a prefab to the Prefabs folder.
- Note: in some cases, you might have to follow a different setup process

To create a toolbox element for the newly created item, follow these steps:

- Import the toolbox texture to the Textures -> Objects -> Toolbox folder as a sprite.
- Drag the texture to the scene to create an object.
- Set the tag to Toolbox.
- Set the sorting layer and layer to GUI.
- Set the sorting order to 1
- Assign a 2D rigidbody and collider(s) to the object
- Assign the ToolboxItemType script to the object.
- Set the width value to (toolboxTexture.width / 100).
- Assign the normal prefab you created to Normal Prefab.
- Drag the toolboxCount2 sprite from Textures -> GUI –> MainScreen to the scene.
- Assign the newly created object to the toolbox object as a child.
- Assign the child to the ToolboxItemType script's Counter position.
- Assign the toolbox numbers.
- Save the item as a prefab to the Prefabs folder.