IDC Robocon 2022

INSTRUCTIONS

Harness the River

August 1st, 2022

CONTENTS

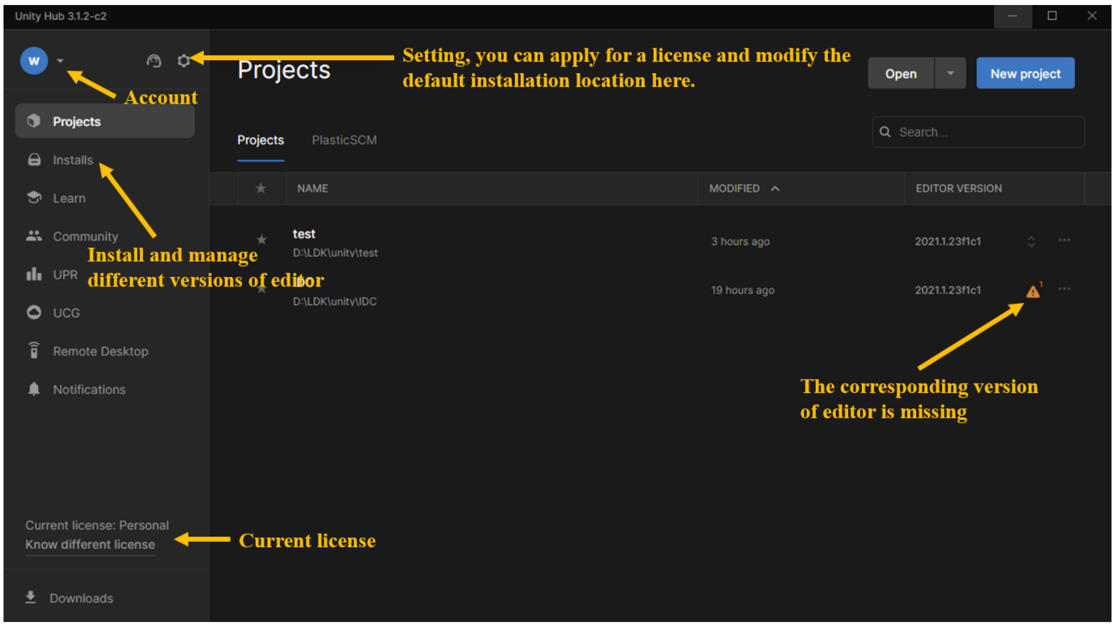
1 Introduction of unity

This chapter will explain unity installation, project creation and package import. The user interface of unity is also described in this chapter. These descriptions will help players get familiar with the development environment and get started quickly. Players can also refer to *Unity User Manual*, *Unity Learn* and *Unity Scripting API*.

1.1 Installation of Unity

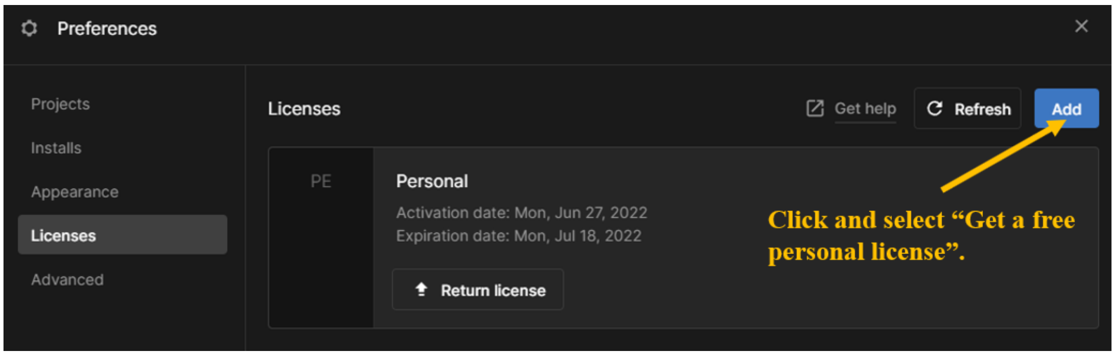
In this game unity 2021.1.23f1c1 is used. Please make sure to install the same version, otherwise the program may run abnormally or even fail to run. Find version 2021.1.23 on the official website of unity for download. It is recommended to install *Unity Hub* to facilitate project management, license update, etc. Generally, you can install the editor according to the default settings. If there is a code editor such as *VSCode* on your computer, you can uncheck the development tool and configure it.

Introduction to *Unity Hub*:

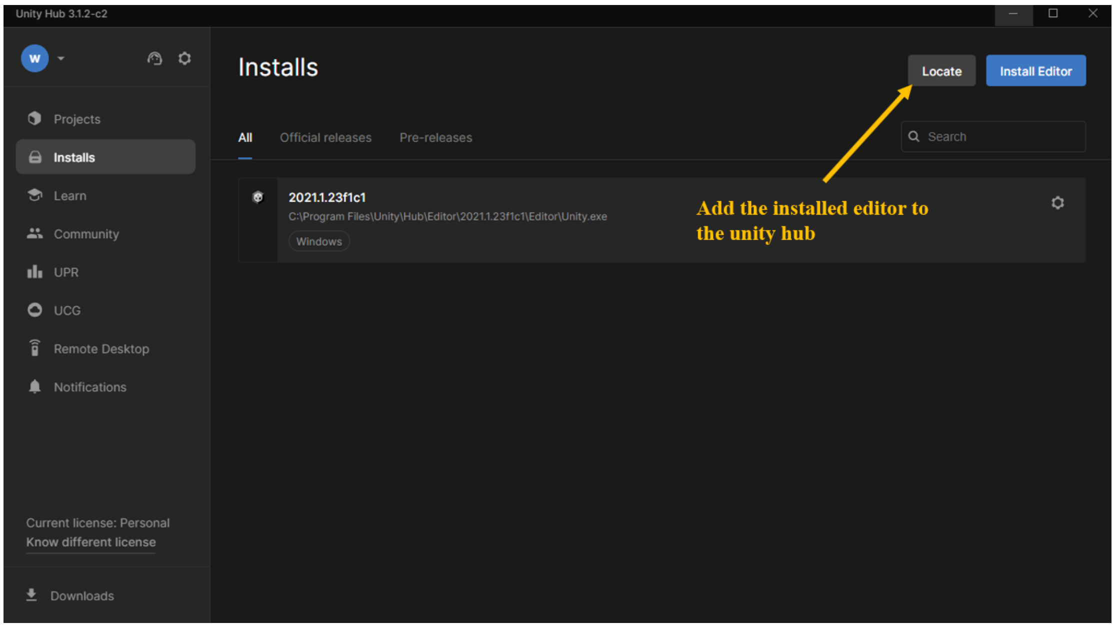


Unity Hub 3.1.2-c2
Setting, you can apply for a license and modify the default installation location here.
Account
Projects
Open
New project
Projects
PlasticSCM
Search...
Installs
Learn
Community
UPR
UCG
Remote Desktop
Notifications
Install and manage different versions of editor
The corresponding version of editor is missing
Current license: Personal
Know different license
Current license
Downloads

	NAME	MODIFIED	EDITOR VERSION
★	test D:\LDK\unity\test	3 hours ago	2021.1.23f1c1
★	IDC D:\LDK\unity\IDC	19 hours ago	2021.1.23f1c1

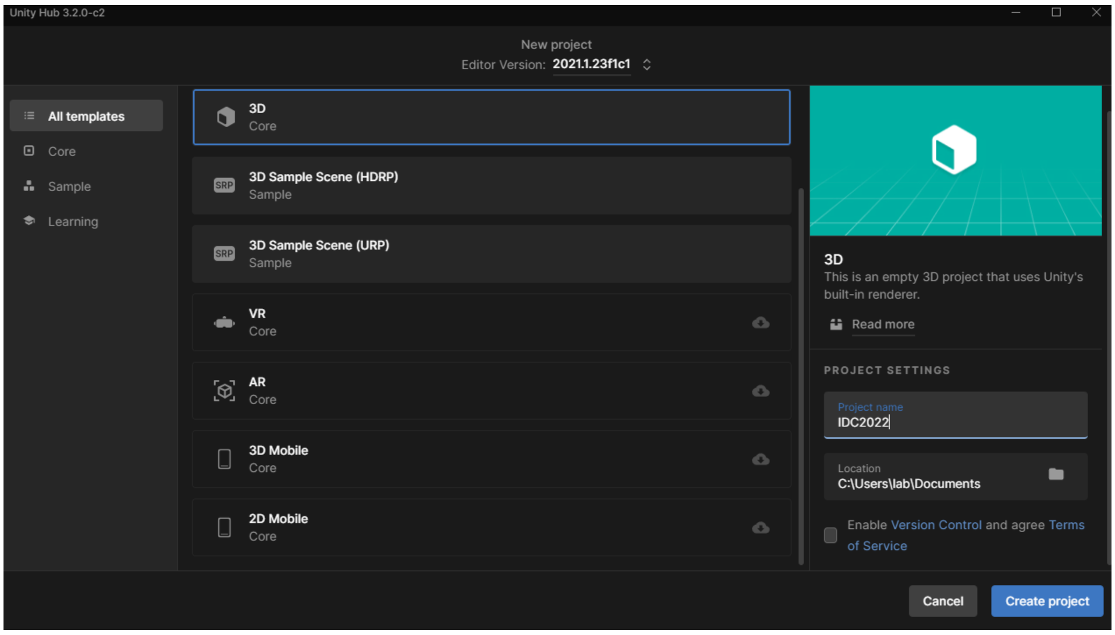


Activation date: Mon, Jun 27, 2022
Expiration date: Mon, Jul 18, 2022



1.2 Project creation and package import

- Click "New project" in Unity Hub and select 3D project to create. Do not select "*Enable Version Control and agree Terms of Service*".

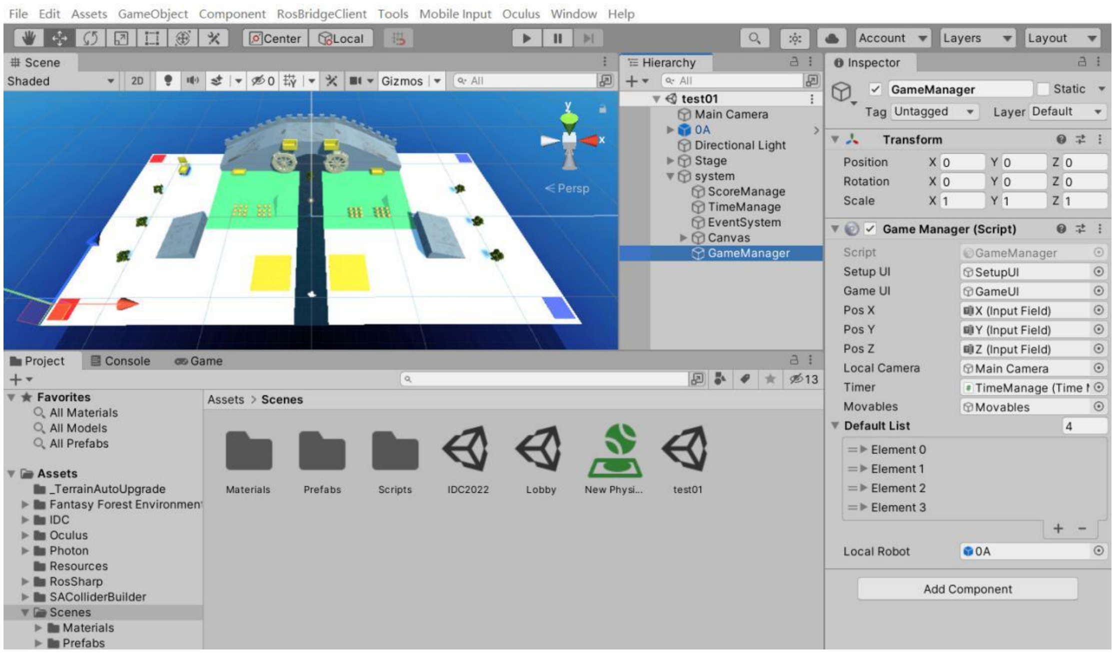


3D
This is an empty 3D project that uses Unity's built-in renderer.

Project name: IDC2022
Location: C:\Users\lab\Documents
☐ Enable Version Control and agree Terms of Service

- Import unitypackage. Select *Assets* → *Import package* → *Custompackage* in the project and select the corresponding unitypackage to import.

1.3 Introduction of interface



GameManager Static
Tag Untagged Layer Default

Transform

Position	X 0	Y 0	Z 0
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

Game Manager (Script)

Script	GameManager
Setup UI	SetupUI
Game UI	GameUI
Pos X	X (Input Field)
Pos Y	Y (Input Field)
Pos Z	Z (Input Field)
Local Camera	Main Camera
Timer	TimeManage (Time
Movables	Movables
Default List	4
	Element 0
	Element 1
	Element 2
	Element 3
Local Robot	0A

- *Project*: Project directory. The file with unity icon is a scene file. Double click it to enter the scene.
- *Scene*: Scene window. By default, the left mouse button is used to select and translate objects; Press and hold the mouse wheel to move the viewing angle; Scroll wheel to zoom the field of view; Press and hold the right left mouse button to rotate the viewing angle. If an object is selected in the hierarchy window, press the F key in the scene to quickly move to the object.
- *Hierarchy*: Hierarchy window, which lists the objects in the scene and their hierarchical relationships. Press the left mouse button can select an object and transfer it with its sub objects to other positions in the hierarchy; Press the right mouse button to create a child object, parent object, modify the activation status, and so on.
- *Inspector*: Inspector window, which allows you to view and modify various components and attributes of the selected object. When setting properties, you can drag prefabs, objects, scripts, etc. from Project directory and Hierarchy window into this window.
- *Console*: displays debugging information.
- *Game*: show the display when the program is running.

The position of these windows can be adjusted. Right click the label to open a new window or close it.

1.4 Common terms

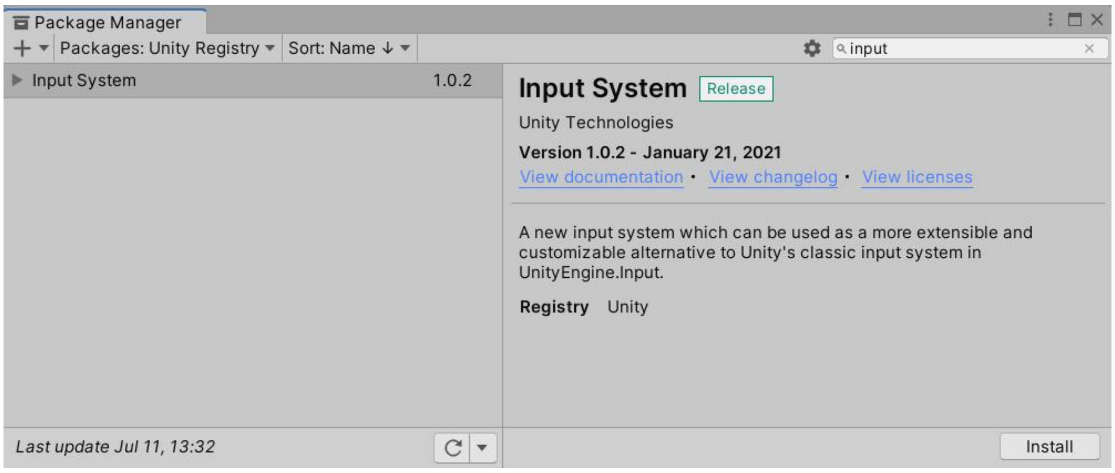
- GameObject: any object in the hierarchy or scene. Empty objects have no impact on the program, and components need to be added to realize various functions. Components can be read through the function Getcomponent< component name > () of the object.
- Transform : a component of every object. In the properties window, it describes the position, rotation and zoom scale of the object relative to its parent object. The child object inherits the transformation of the parent object and moves with the parent object.
- Rigidbody: basic physical component. The motion of the object to which rigidbody is added will only be affected by physical effects such as gravity, and will not automatically move with the parent object.
- Collider: a basic physical component. The object to which collider is added can collide with other objects with colliders. The collision range is determined by the geometric contour of the collider. If *IsTrigger* is checked, other objects will not collide with the object, but an event can be triggered when it contacts the collider. This kind of collider is generally called a trigger. The collision body of the parent object is the union of its child object's collision bodies, also known as composite collision bodies.
- Wheel Collider: a special collider component, which has a special mechanical model and mechanical parameters similar to rigid bodies such as mass and friction. It is recommended to learn more from official manual.
- Joint: used with rigid bodies to connect different rigid bodies with each other. In fact, it is not a rigid connection, but similar to a spring. If the two ends of the connection are subjected to a great force, they may move out of the limited range. If you do not specify the rigid body connected at the other end of the joint, the object to which the joint is added will be anchored in the world coordinate system by default.
- Script: the code that controls the behavior of the object. By default, it belongs to the Monobehavior class. It takes c# as the programming language and contains the start() and update() functions, which run at the first frame and every frame after the object is activated. See the Manual and Scripting API for other common classes and their usage.
- Prefab: a prefab is a "template" object. Dragging it into the scene can create an instance exactly the same as the prefab. The modification of the prefab will be automatically synchronized to all instances of the prefab at the same time. Therefore, using the prefab can quickly create a large number of identical objects.

2 Instructions for the provided package

2.1 Project Instructions

Download *unitypackage* from:
https://cloud.tsinghua.edu.cn/f/71a739165733449 48e4c/
Configuration:

- Click "New project" in Unity Hub and select 3D project to create.
- Import the *unitypackage* provided (*Assets* → *Import Package* → *CustomPackage*),
- Install the *Input System* package: In *Window* → *Package Manager*, change the list to *Packages: Unity Registry* in the upper left corner to list all the packages, then search the *Input System* in the search box in the upper right corner. Finally select the package and click the button in the lower right corner to install.



Unity Technologies

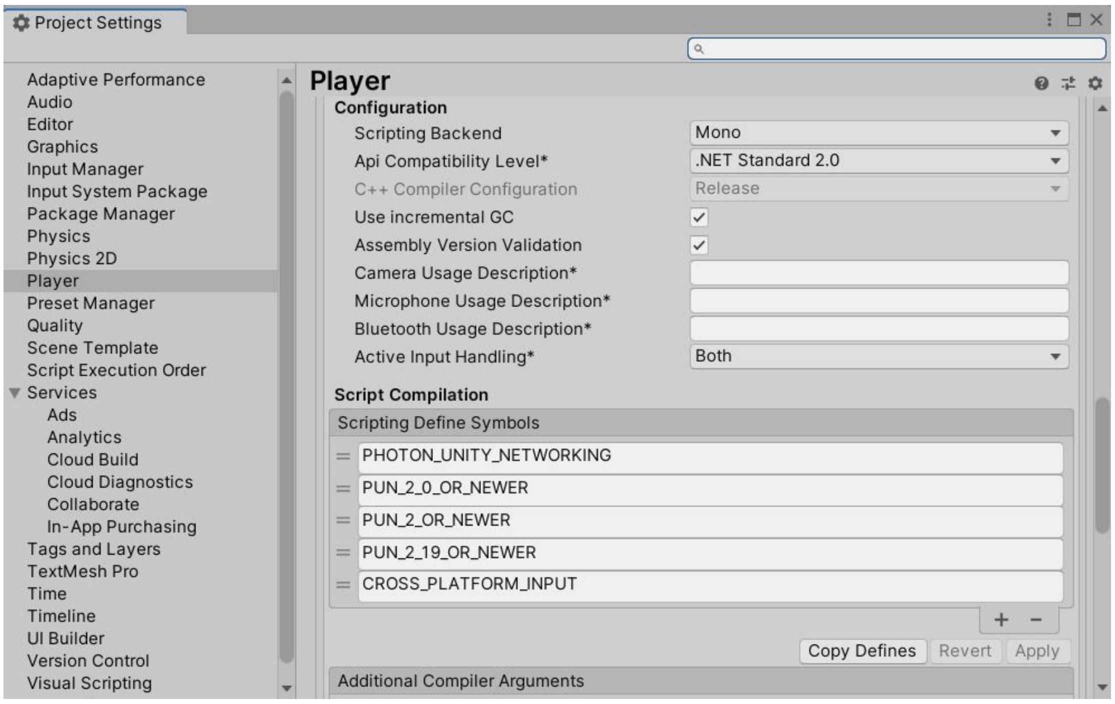
Version 1.0.2 - January 21, 2021

A new input system which can be used as a more extensible and customizable alternative to Unity's classic input system in UnityEngine.Input.

Registry Unity

Last update Jul 11, 13:32

- Adjust packaging settings: In *File* → *Build Settings*, click *Add open scenes* to add the currently open scenes. Use this method to add *Lobby* and *IDC2022*.
- Adjust input settings: In *Edit* → *Project Settings* → *Player*, find *Other Settings* → *Configuration* → *Active Input Handling* and select *Both*. This setting may not be saved after the editor restarts, so it needs to be checked.



Player

Configuration

Scripting Backend	Mono
Api Compatibility Level*	.NET Standard 2.0
C++ Compiler Configuration	Release
Use incremental GC	✓
Assembly Version Validation	✓
Camera Usage Description*	
Microphone Usage Description*	
Bluetooth Usage Description*	
Active Input Handling*	Both

Script Compilation

Scripting Define Symbols

```
PHOTON_UNITY_NETWORKING
PUN_2_0_OR_NEWER
PUN_2_OR_NEWER
PUN_2_19_OR_NEWER
CROSS_PLATFORM_INPUT
```

Additional Compiler Arguments

Folders that contestants are mainly concerned with are *Scenes* and *Resources*.

Scenes is the main folder that contains the scenes, prefabs, scripts and other things used in the contest. *Resources* is the folder that Photon Unity Networking (PUN) uses for instantiating objects. Things created in the last IDC contest are scattered in the *IDC* folder and *Assets* folder. Only part of them are removed, and you may use the rest of them for reference. The rest of the folders contain packages, scripts and other things needed to support the program.

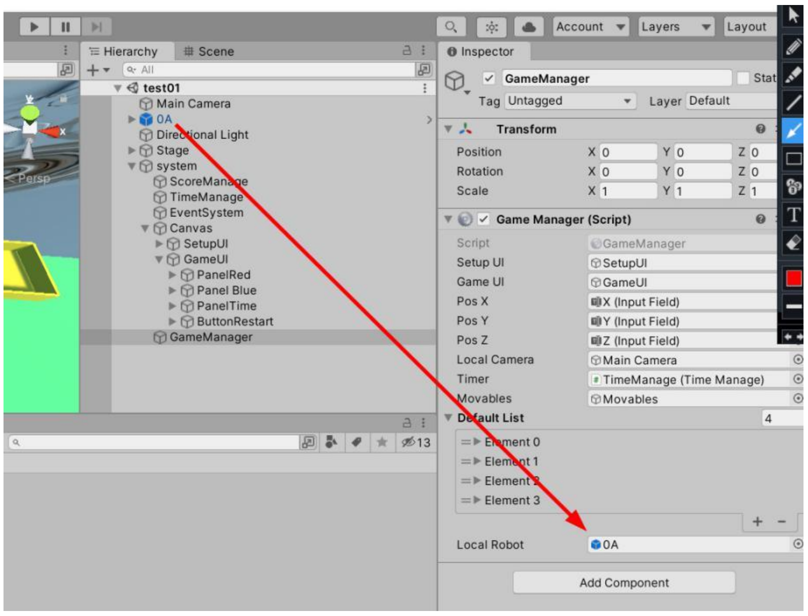
The *Script* folder contains 2 subfolders. The *System* folder includes scripts that support gameplay. Scripts outside the *System* folder are provided for the participants to use, but you should not change the code inside the scripts. If you need to make changes, create a new script. It is recommended to put the new scripts you have created into the *PlayerScript* folder to keep the project tidy.

The *Prefabs* folder contains the prefabs. To create a prefab simply drag the gameobject from the *Hierarchy* window to the *Project* window. Like the *Script* folder, there is a *PlayerPrefabs* folder for you to store your prefabs.

When naming your prefabs and scripts, it is recommended that the name starts with *your team number + A* or *your team number + B*, so as to avoid having the same name as the creation of other teams.

Double click *test01* in the *Scenes folder* to enter the field, where you can build and test your robot. Select *GameManager* in the *Hierarchy* window, drag your robot from the *Hierarchy* window to *LocalRobot* in the *Inspector* window to finish configuration.

When directly running this scene you can not adjust your starting position run-time, and you will not be timed. In an actual match you will start from the *Lobby* scene. In this case you need to put your robot prefab into the *Resources* folder in advance.



GameManager
Tag Untagged Layer Default

Transform
Position X 0 Y 0 Z 0
Rotation X 0 Y 0 Z 0
Scale X 1 Y 1 Z 1

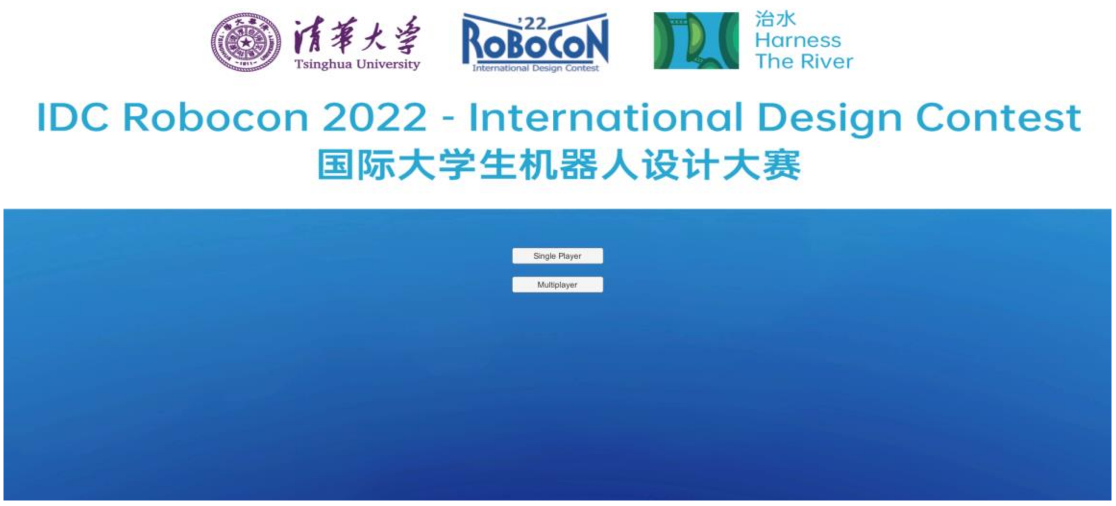
Game Manager (Script)
Script: GameManager
Setup UI: SetupUI
Game UI: GameUI
Pos X: X (Input Field)
Pos Y: Y (Input Field)
Pos Z: Z (Input Field)
Local Camera: Main Camera
Timer: TimeManage (Time Manage)
Movables: Movables
Default List: 4
Element 0
Element 1
Element 2
Element 3
Local Robot: 0A

The *IDC2022* scene is a backup duplication of *test01* for multiplayer cases. **Do not make any changes to it**, or you may not be able to connect with others.

2.2 Instructions for running this program

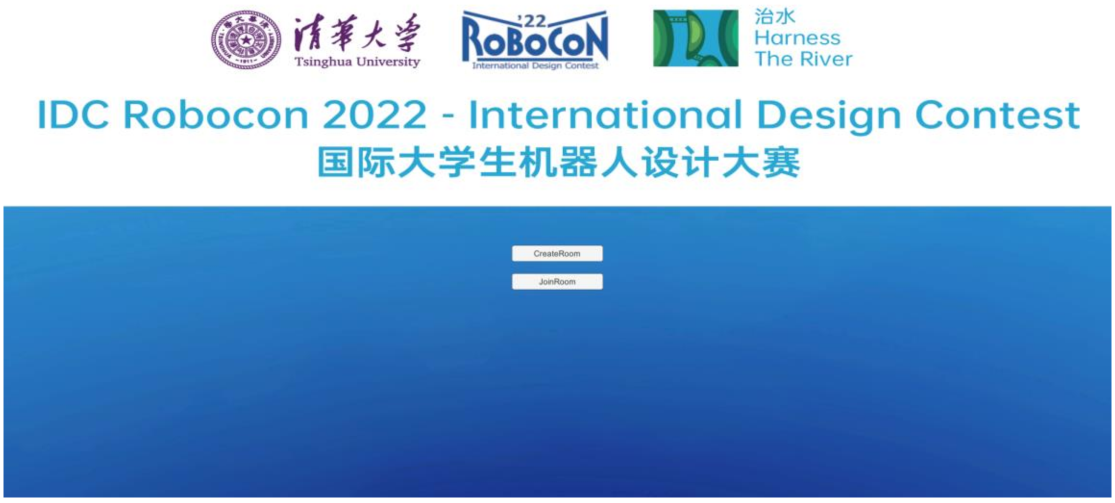
Click *Build And Run* to make a build of the program.

Double click *New Unity Project.exe* (Depending on how you name this program) to open the program and enter the lobby scene. Single player mode can be run offline.

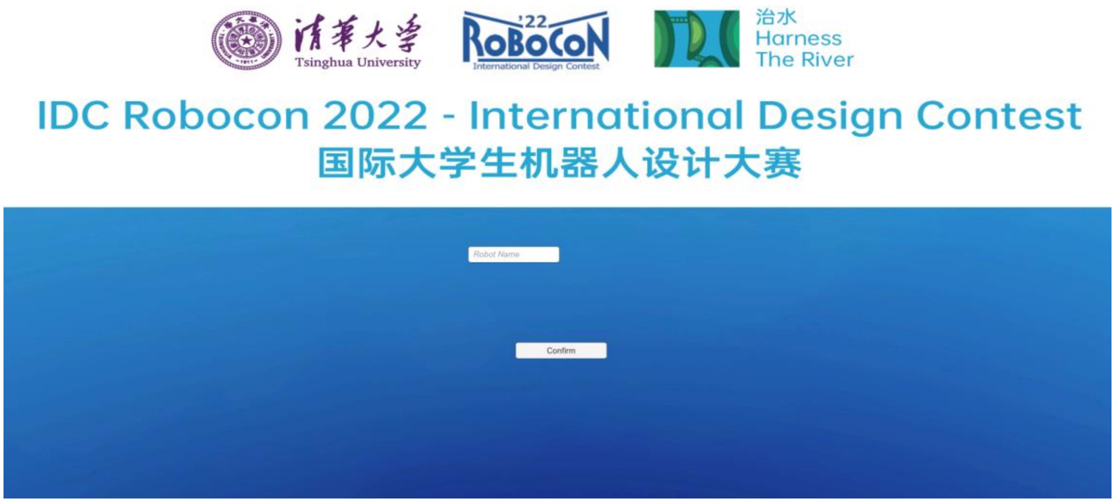


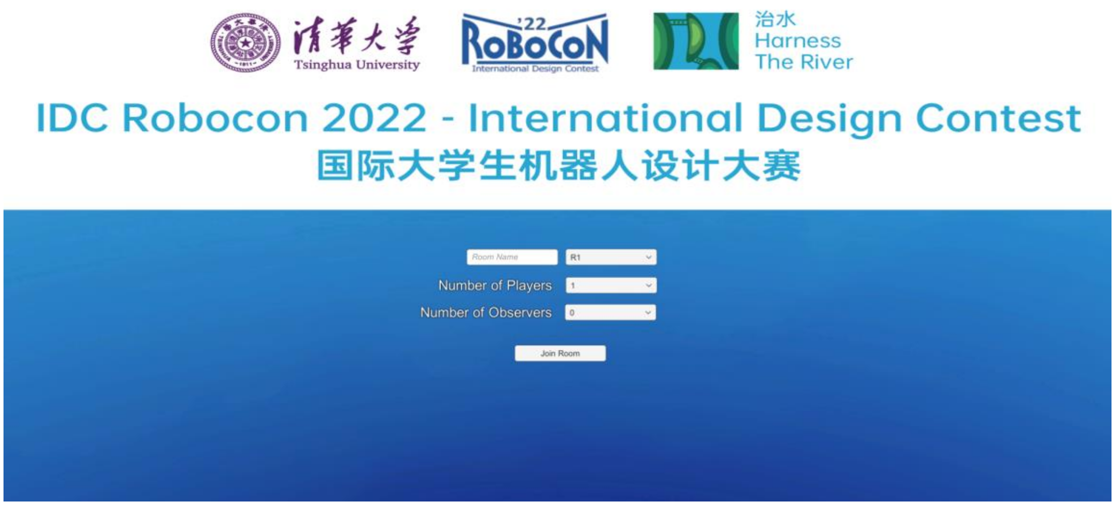
Click the “*Multiplayer*” button, the program will connect to the server first and “Connecting” will appear on the interface. After that, two buttons "*CreatRoom*" and

"*JoinRoom*" will appear.



Click the "*CreateRoom*" button to enter the room setting interface. First you need to select the robot used by the host. Initially the program includes robots such as *0A* and *0B*. Before creating a build, all robots used in the match should be included in the *Resources* folder. If the host is a player in this match, you need to enter the name of a prefab such as "*0A*". If the host is not a player but an observer, you can enter anything here since the observer don't need to own and control a robot. Press the "*Confirm*" button for more settings.

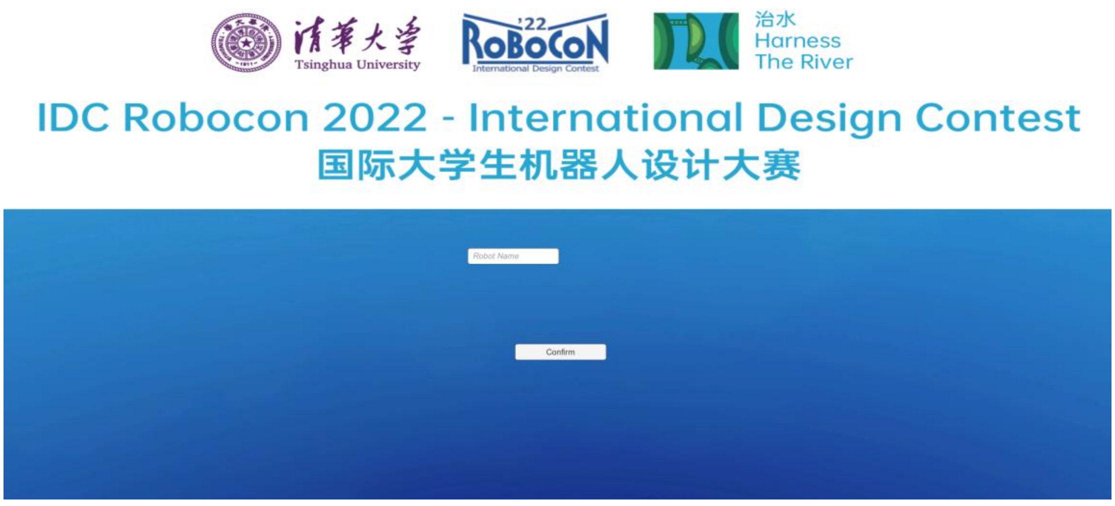




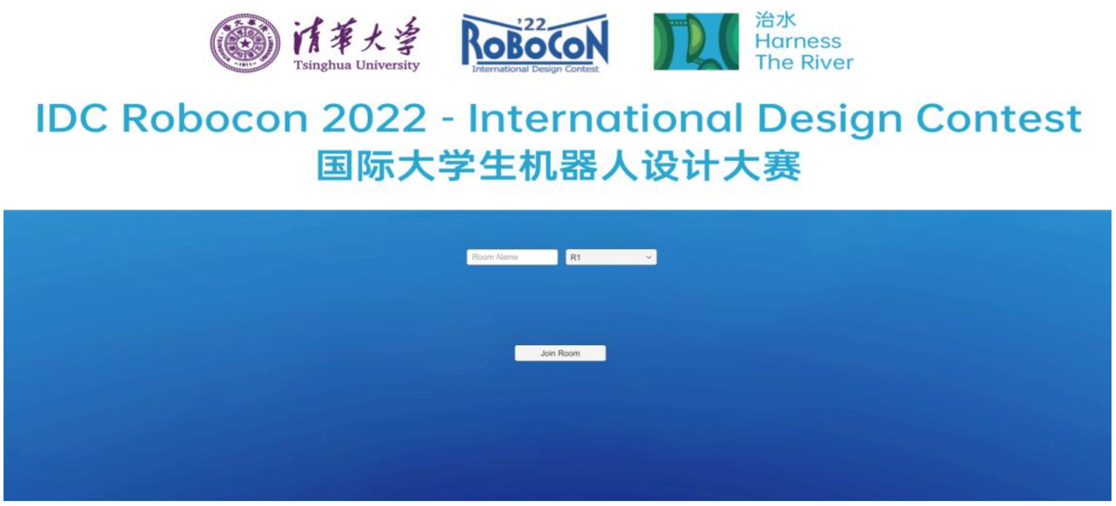
In the interface shown in the above figure, you need to type in the room name, select a role that the host acts and set the number of players and observers. Note that the game will only start when the number of players and observers in the room equals to the number you set and everyone is ready.

Before the contest, room will be created by the IDC staff, and players only need to join the room.

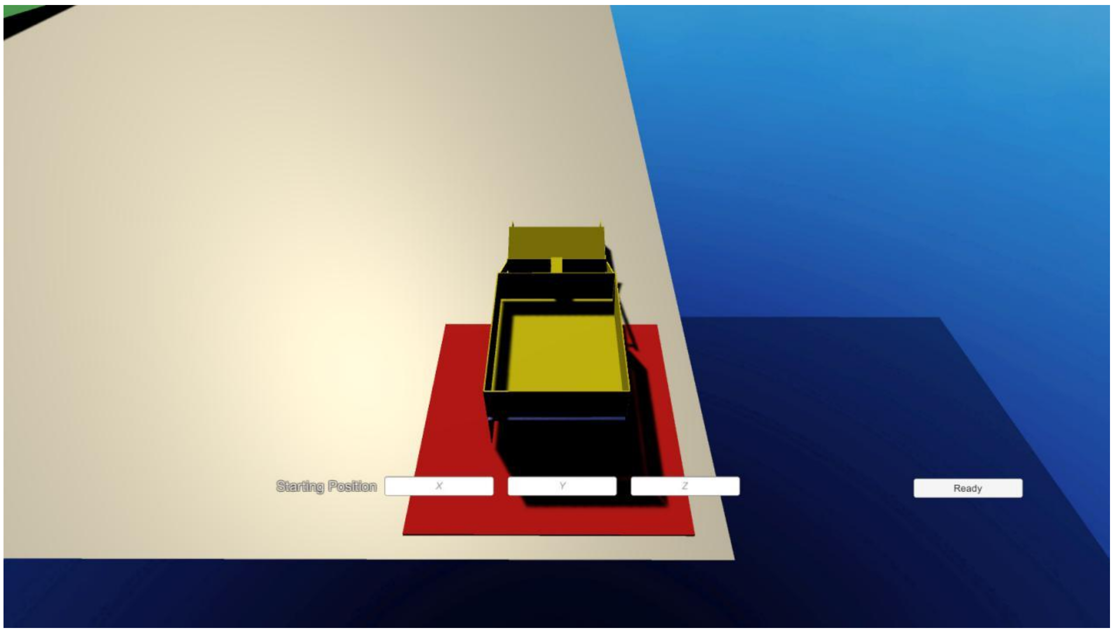
After clicking the "*JoinRoom*" button, a new page for players to select their robots will appear. In the input field enter the name of your robot prefab. Then click the confirm button and go to room settings.



In room settings, players need to type in the room name and the select roles. The room name will be announced before each match. There are totally six roles to choose. "OB1" and "OB2" are two observers. In the contest, only the IDC staff can choose the two roles. Players need to choose from "R1, R2, B1, B2". R1 means starting area 1 on the red side, which is located on the upper left corner of the field with the task of ball transportation. R2 means starting area 2 on the red side, with the task of planting trees. Due to the limitations of PUN, interacting with objects of other participants may cause serious lagging (especially the balls, which bump into each other frequently since they are densely packed), so make sure that you choose the correct starting area. Click *Join Room* to confirm your settings and join the room.

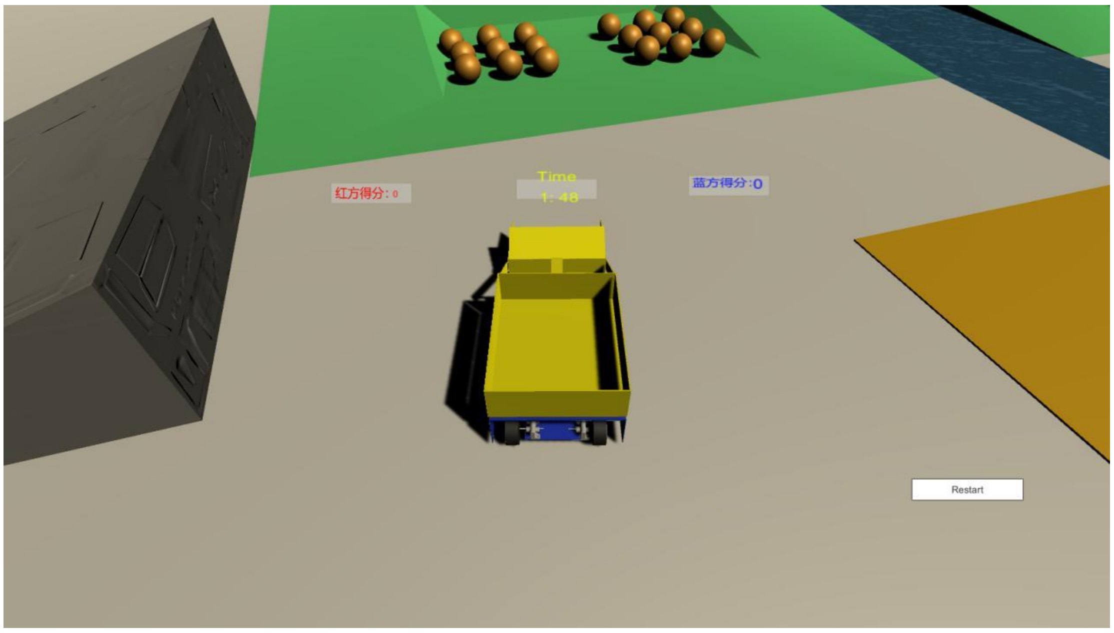


On joining the room your robot will appear on the chosen starting area. By entering new coordinates in input fields X, Y or Z, you can adjust your starting position, but your robot can't go beyond the starting area. Click *Ready* when you are finished. The match will start after all participants are ready. If you cannot see the button and the input fields, enlarge the window size.



When the match starts, the timer and the score of both teams will appear. Click Restart to return your robot to its starting position.

When you are controlling your robot, you can press “C” and “V” on the keyboard to switch between fixed view and free view. The angle of the camera will change as the mouse moves in free view.



2.3 Introduction of the Built-in Robots

Control the movement of the robots with arrow keys or *w, a, s, d*.

0A:

A robot with a shovel and a dumper. Use *j* and *k* to flip the dumper, while *m* and *n* are used to control the shovel.

0B:

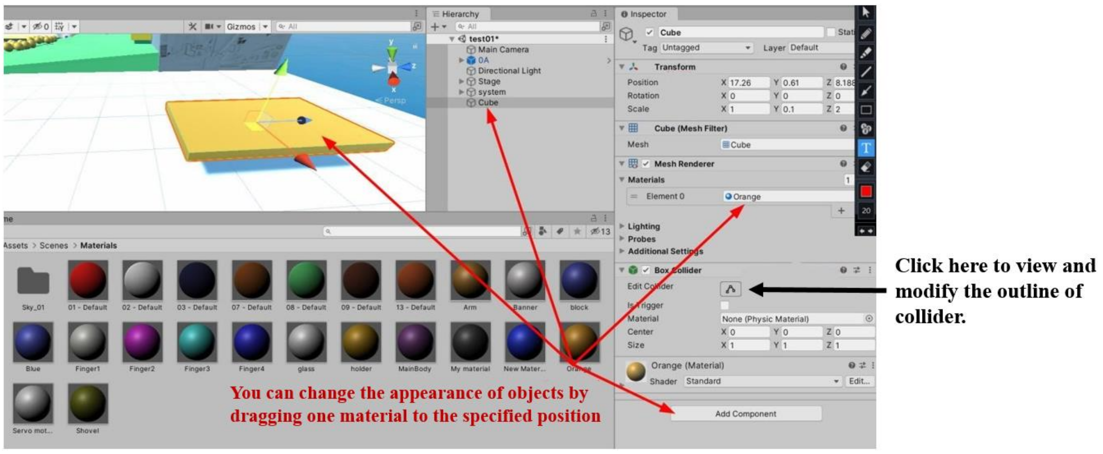
A robot with a pincer. Use *o* and *p* to open and close the pincer.

3 Instructions for Robot construction

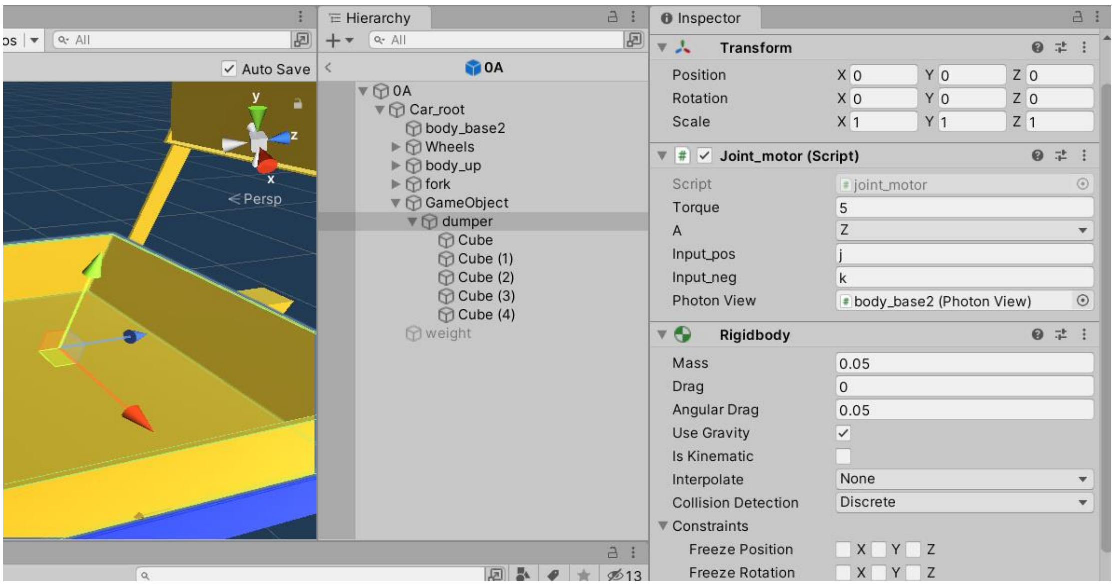
This chapter will explain how to construct a robot, including modeling with basic geometry in unity, setting joints, adding scripts, adjusting wheel colliders, etc. In terms of modeling, you can create models in an external application such as SolidWorks, and then import them into Unity. you can also install Probuilder to model complex geometry. However, it should be noted that complex models and complex collider will make the calculation time longer.

3.1 Modeling

Right click in the hierarchy window and select base geometry in *3D Objects*. These geometries contain collider when they are created, and generally it doesn't need to be changed. Basic geometry of unity includes plane, but the plane is only visible on one side, and the back is transparent. Therefore, the cube should be used as a flat plate. Create a cube, adjust the scale in its transformation component, and turn it into a rectangular plate as the chassis of the robot. Find *Scenes* → *Materials* in *Project*. Select one material and drag it to *Properties* of the cube to change its appearance. Click *Add Component* in *Inspector*. Then search and add *Rigidbody*.


You can change the appearance of objects by dragging one material to the specified position
Click here to view and modify the outline of collider.

Let's take dumper as an example to show the modeling of complex structure. The dumper consists of five plates. Right click the cube, select *Duplicate* to copy it, and then adjust the transformation of the new cube. The modeling of complex object does not require adding *Rigidbody* to each component and then splicing them. In this way, all parts of the dumper can't be contacted at all, otherwise there will be collision and disassembly during actual operation. If you keep all the parts at a distance, a large number of joint components are needed to connect these rigid bodies with each other. The process is cumbersome and the connection is not firm, so it is not recommended. A better way is to create an empty object "dumper" first, and take each cube as the sub object of the "dumper". Then the collision body of the "dumper" is the composite collider formed by the colliders of all the five cubes. After that, add *Rigidbody* to the "dumper". In this way, the "dumper" is an independent physical entity, which is firmly connected and easy to control.



0A
- Car_root
  - body_base2
  - Wheels
  - body_up
  - fork
  - GameObject
    - dumper
      - Cube
      - Cube (1)
      - Cube (2)
      - Cube (3)
      - Cube (4)
    - weight

Transform

Position	X 0	Y 0	Z 0
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

Joint_motor (Script)

Script	joint_motor
Torque	5
A	Z
Input_pos	j
Input_neg	k
Photon View	body_base2 (Photon View)

Rigidbody

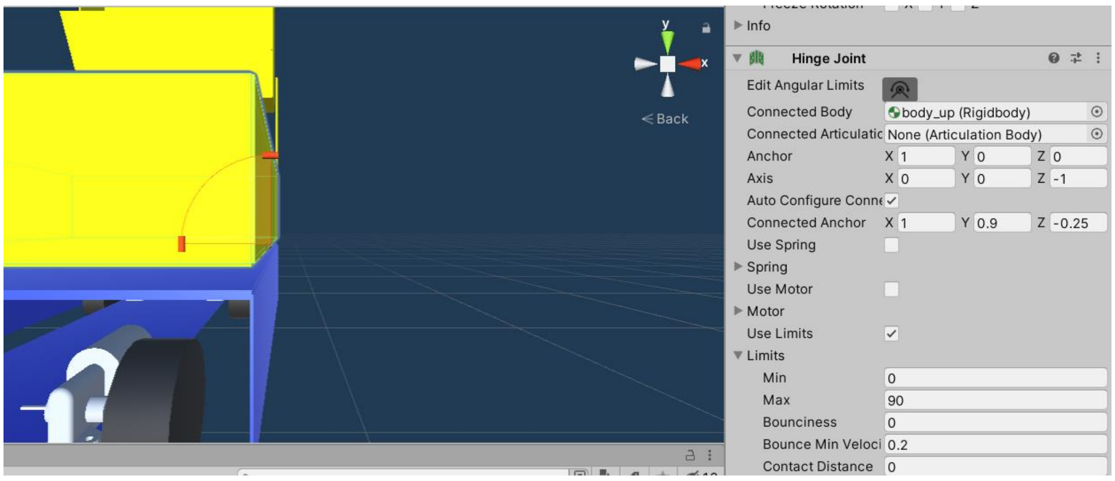
Mass	0.05
Drag	0
Angular Drag	0.05
Use Gravity	✓
Is Kinematic	
Interpolate	None
Collision Detection	Discrete
Constraints	
Freeze Position	X Y Z
Freeze Rotation	X Y Z

3.2 Add and configurate joints

Add *Hinge Joint* to the dumper in *Inspector* so that the dumper can rotate around a specific axis. Unity uses the left-handed coordinate system, and the coordinate of vector is (x, y, z). The corresponding relationship between the three axes and the direction is:

Front → positive direction of Z axis; Up → positive direction of Y axis; Right → positive direction of X axis.

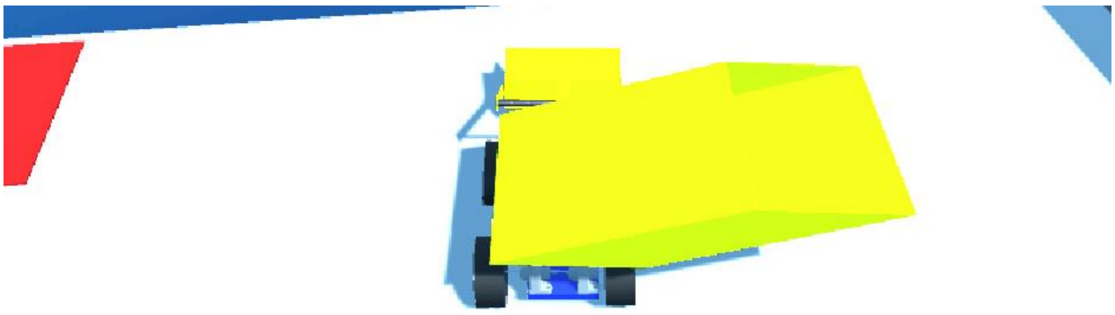
Suppose you want to dump the ball from the right side of the car. Then set the *Anchor* on the right lower edge of the dumper. Set the direction of the rotation axis to be the negative direction of the Z axis, so that the positive direction of rotation is clockwise. Tick *Use Limits* and set the maximum angle to 90 °. You can also click *Edit Angular Limits* and drag the terminals at both ends of the sector to modify it. Finally select other objects to which the object is connected through joints in *Connected Body*.



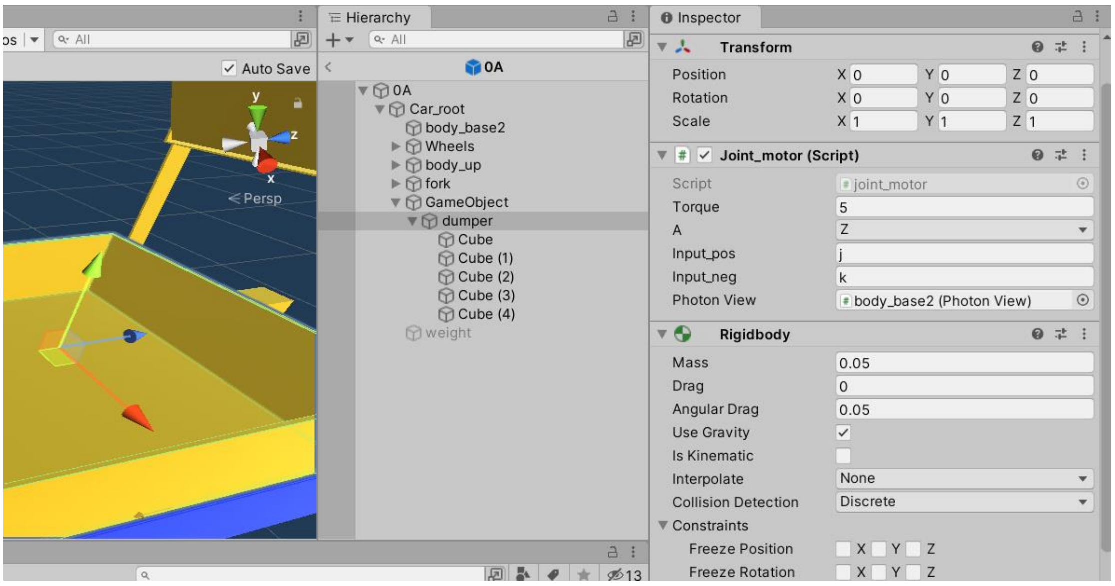
Hinge Joint

Edit Angular Limits			
Connected Body	body_up (Rigidbody)		
Connected Articulatic	None (Articulation Body)		
Anchor	X 1	Y 0	Z 0
Axis	X 0	Y 0	Z -1
Auto Configure Conne	✓		
Connected Anchor	X 1	Y 0.9	Z -0.25
Use Spring			
Spring			
Use Motor			
Motor			
Use Limits	✓		
Limits			
Min	0		
Max	90		
Bounciness	0		
Bounce Min Veloci	0.2		
Contact Distance	0		

Note that changing the attributes of an object in unity will not directly adjust its size, but its scale in the X, Y, and Z dimensions. If the simplest scale of the object is not 1:1:1, the shape will be distorted due to scaling during rotation. For example, if the width, that is, the proportion of the x-axis, is significantly enlarged, the x-axis will correspond to the height of the dumper during the rotation around the z-axis, which will lead to an abnormal increase in the height of the dumper during the rotation.



The child object will inherit the transform of its parent object. When one object is moved in *Hierarchy* to become the child object of other objects, unity will automatically adjust its zoom scale to maintain the size of the object in the world coordinate system. This adjustment is easy to cause the distortion described above. To avoid this, right-click the object in the *Hierarchy* and select "*Create Empty Parent*" to create an empty parent object for it, so that the scale of the empty object will be modified when moving, while the scale of the real object to be controlled remains 1:1:1.



```
0A
▼ 0A
  ▼ Car_root
      body_base2
    ▶ Wheels
    ▶ body_up
    ▶ fork
    ▼ GameObject
      ▼ dumper
          Cube
          Cube (1)
          Cube (2)
          Cube (3)
          Cube (4)
      weight
```

Transform

Position	X 0	Y 0	Z 0
Rotation	X 0	Y 0	Z 0
Scale	X 1	Y 1	Z 1

Joint_motor (Script)

Script	joint_motor
Torque	5
A	Z
Input_pos	j
Input_neg	k
Photon View	body_base2 (Photon View)

Rigidbody

Mass	0.05
Drag	0
Angular Drag	0.05
Use Gravity	✓
Is Kinematic	
Interpolate	None
Collision Detection	Discrete
Constraints	
Freeze Position	X Y Z
Freeze Rotation	X Y Z

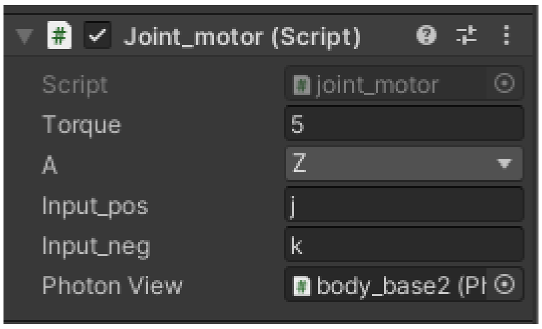
3.3 Motor and synchronizing assembly

Most actuators in this contest are provided in script form, including:

- *Drive Wheel L/R* (Bosh motor)
- *Joint_motor* (DC motor)
- *Pump* (Air Cylinder)

- *DroneCtrl, PropellersCtrl, PropellerCtrl* (Sky Engine)

You can directly add scripts to the objects you want to drive to achieve the corresponding functions. For example, *Drive Wheel L/R* is added to the car's wheel to control the car's movement. *Joint_motor* is added to the dumper to control its rotation. *Pump* is added to *0B*'s *fingerL/R* to control their translation to clip the tree. Using *DroneCtrl, PropellerCtrl* you can design a quad-rotor aircraft. Let's take *Joint_motor* as an example. You can select the axis of rotation in the menu *A*. The axis of rotation refers to the coordinate system of the object itself (that is, the coordinate system that appears after the object is selected). You can set the keys to control the positive and negative input in *Input_pos* and *Input_neg*.


Joint_motor (Script)

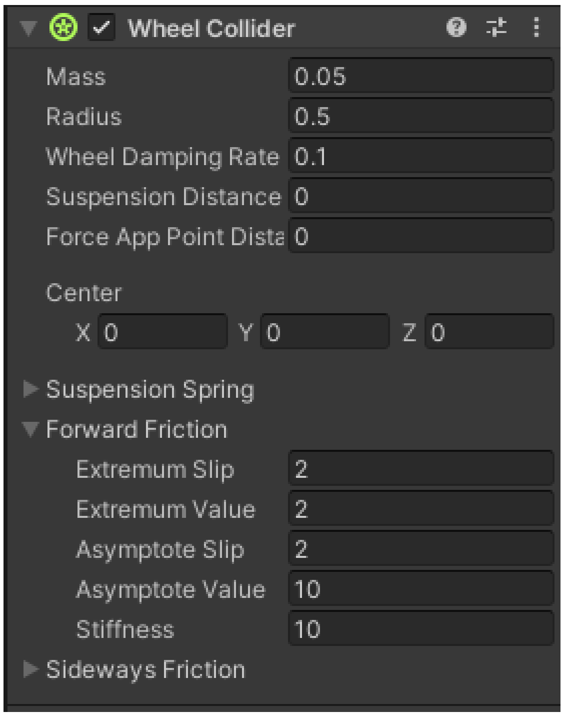
Script	joint_motor
Torque	5
A	Z
Input_pos	j
Input_neg	k
Photon View	body_base2 (Pl

Please refer to the manual for key names corresponding to different keys. For more details about the Sky Engine, please refer to the appendix at the end of this instruction. A complete UAV named *drone* is provided in the package. You are allowed to install custom parts under the fuselage to implement various functions.

Objects that accept input and interact with the outside world, such as arms, dumpers, wheels, and the whole robot, need to be synchronized to avoid accepting input from other clients and synchronize the interaction to other clients in time. Common components for synchronization include *Photon View*, *Photon Transform View* and *Photon Rigidbody View*, which are used to observe other synchronized objects, the transformation of synchronized objects and the stress condition of synchronized objects respectively. Add the above three components to the object which needs to be synchronized.

3.4 Adjustment of *Wheel Collider*

Wheel Collider is a physical component developed by unity specifically for simulating wheels. Different from the name, *Wheel Collider* includes not only *Collider*, but also physical parameters such as mass, friction and damp, as well as parameters such as spring and damping of suspension. There is no need to add *Rigidbody* to achieve physical effects. The calculation of *Wheel Collider's* friction is independent of other physical engine, that is, it is not affected by the physical materials of other objects. To change the grip performance of the wheels, it is necessary to adjust the stiffness of the *Forward Friction* and *Sideways Friction* in the *Wheel Collider*. Please refer to the manual for the meanings of each coefficient.


Wheel Collider

Mass	0.05
Radius	0.5
Wheel Damping Rate	0.1
Suspension Distance	0
Force App Point Dista	0

Center

X 0 Y 0 Z 0

Suspension Spring

Forward Friction

Extremum Slip	2
Extremum Value	2
Asymptote Slip	2
Asymptote Value	10
Stiffness	10

Sideways Friction

Increasing the *Wheel Damping Rate* of the wheel collider can control the shaking and vibration of the car, but it will also reduce the speed accordingly. The higher the *Spring* value in the *Suspension Spring*, the faster the wheel can return to the normal position, and the greater the elasticity; The higher the *Damper* value, the stronger the buffer. The mass of the wheel will also affect the elastic force generated during the collision.

Since the wheel collider is a component rather than an object, the wheel is usually visualized with a cylinder. It is necessary to adjust the radius of the *Wheel Collider* to make it larger than the collider of the cylinder itself, so that it can touch the ground, otherwise the wheel collider will not work.

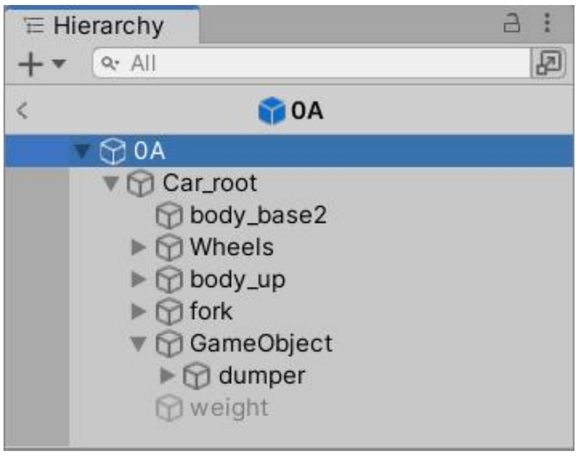
3.5 Make your robot more stable

The rigid objects of the robot are connected through joints, but they are not completely firm. Anomalies such as shaking may appear when these objects are under force. If the robot passes through the mold of other rigid bodies when shaking and moving, the robot may even disintegrate or lose control. You can use the following measures to alleviate this problem:

- Increase the thickness of each part and the distance between each part;
- Add multiple identical joints to the same object to enhance the strength of the connection;
- If the abnormality is caused by collision with a specific object (such as a ball in the field), add a baffle on the robot to avoid such collision;
- In *Edit→Project Settings*, change the *Solver Type* used by the physics engine to *Temporal Gauss-Seidel(TGS)* to reduce joint jitter.

3.6 Object management in *Hierarchy*

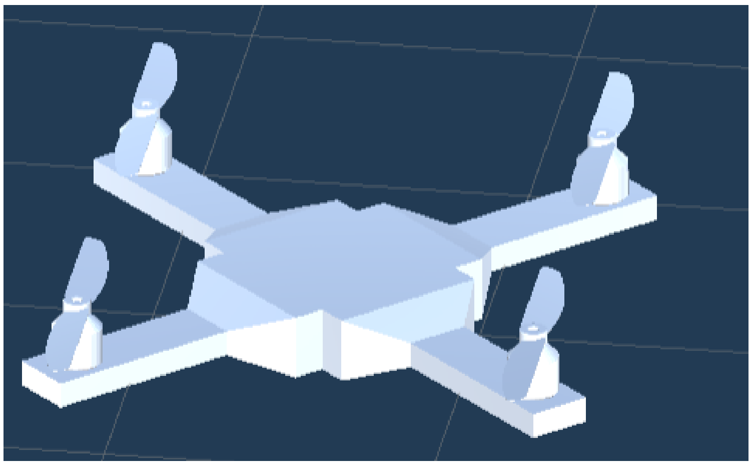
From the previous study, we can know that not all objects need to add rigid bodies, motors, synchronous components, etc., and some components such as rigid bodies may even appear abnormal when they are not added properly. When designing a robot, you can first plan different parts (such as chassis and manipulator) and create corresponding objects, and then use sub objects to build them respectively, which can facilitate the management of each part and avoid this anomaly.

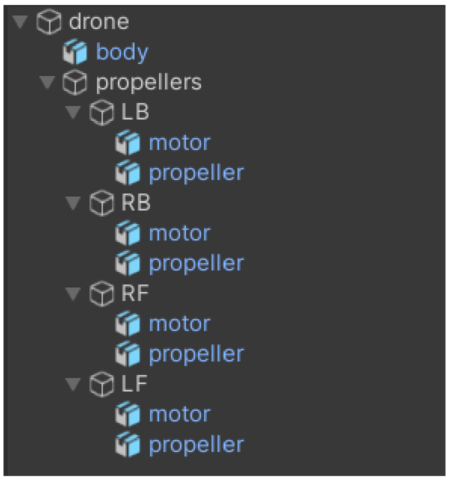


Appendix 1: Instructions for the UAV "*drone*"

A. Structure

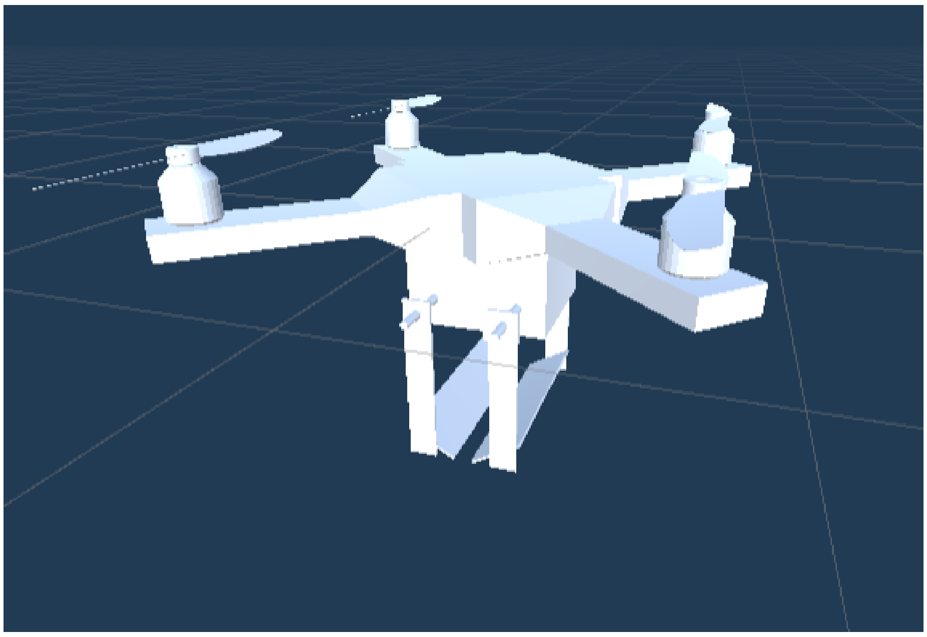
UAV *drone* consists of fuselage and wing. The wing part includes four motors and propellers, which are located at the left front, left rear, right front and right rear respectively.





```
drone
  body
  propellers
    LB
      motor
      propeller
    RB
      motor
      propeller
    RF
      motor
      propeller
    LF
      motor
      propeller
```

The structure of the UAV itself cannot be modified. Players can install custom components under the fuselage to complete the task. For example:



B. Description of the script

Each propeller contains a *PropellerCtrl* script for single propeller control. You are not allowed to modify this script. The script will be automatically called by the upper script.

The whole wing contains a *PropellerCtrls* script, which is used to control all the propellers and simulate the torque generated by four propellers. You are not allowed to modify this script. The script contains an interface function:

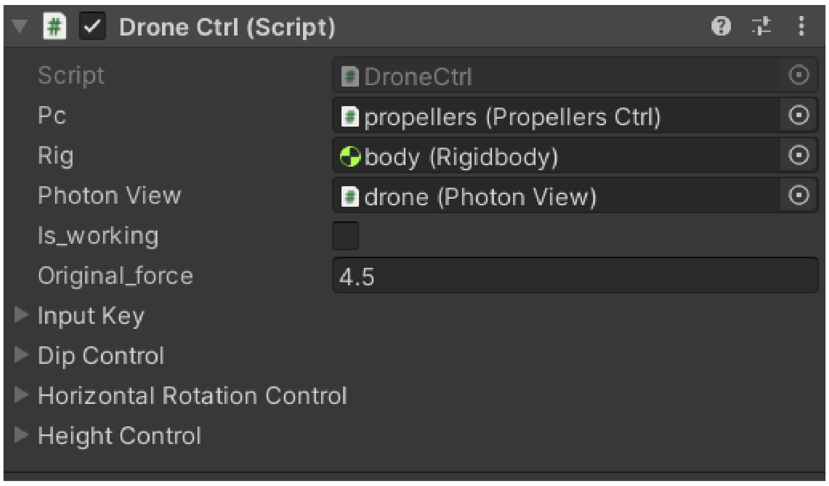
```
public void SetForce(float Inuput_LFforce, float Inuput_LBforce, float Inuput_RFforce,
float Inuput_RBforce)
```

which is used to set the lift of four propellers.

A *DroneCtrl* script is attached to the UAV to control its movement. This script allows players to make modifications, but functions that directly change the object transform cannot be used in the script. You are allowed to modify this script, but functions that directly change the object's transform cannot be used in the script.

The provided *DroneCtrl* script is a reference scheme for controlling the UAV. You can directly use this script as the control script of the UAV, or rewrite it for better performance.

C. Instructions for the *DroneCtrl* script

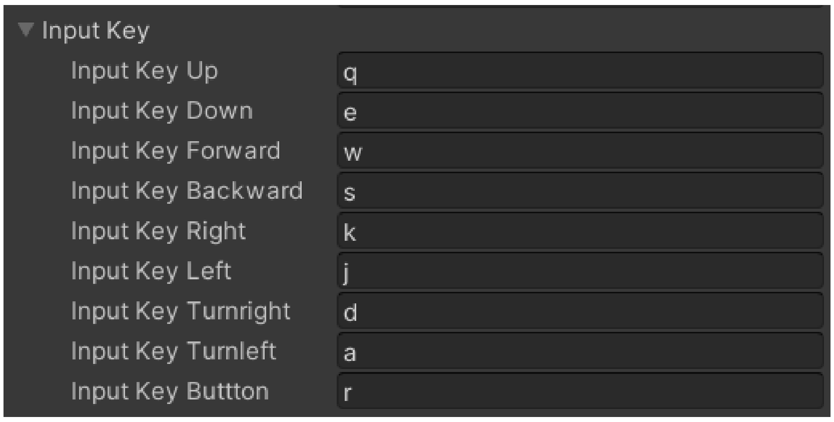

Drone Ctrl (Script)

Script	DroneCtrl
Pc	propellers (Propellers Ctrl)
Rig	body (Rigidbody)
Photon View	drone (Photon View)
Is_working	☐
Original_force	4.5
Input Key	
Dip Control	
Horizontal Rotation Control	
Height Control	

- *Pc*: Bind the corresponding wing control script (*PropellersCtrl*).
- *Rig*: Bind the *Rigidbody* of the fuselage.
- *Photon View*: Bind the UAV's *Photon View* script.

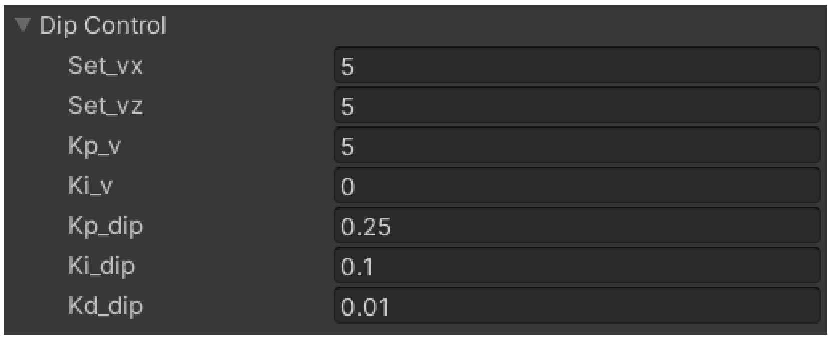
There is no need to modify the above three parameters.

- *Is_working*: Displays the on / off status of the aircraft. This part will be automatically modified as the player operates.
- *Original_force*: The lift provided by a single propeller when the UAV is balanced. The recommended value is the UAV's gravity (including the components installed by the players) divided by 4. The gravitational acceleration is taken as $9.81m/s^2$.
- *Inputkey*: Set keys to control the UAV. From top to bottom: move up, move down, move forward, move backward, move right, move left, turn right, turn left, power on / off.


Input Key

Input Key Up	q
Input Key Down	e
Input Key Forward	w
Input Key Backward	s
Input Key Right	k
Input Key Left	j
Input Key Turnright	d
Input Key Turnleft	a
Input Key Buttton	r

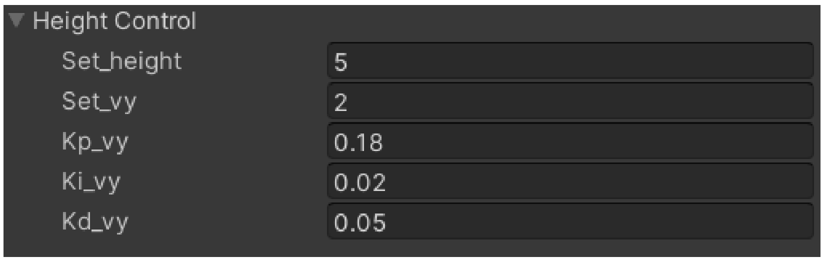
- *DipControl*: Inclination control. Because each propeller can only provide axial lift, the movement of the UAV needs to be realized by the tilt of the fuselage. Inclination control also corresponds to speed control.
  Set_vx: Set the maximum lateral movement speed (m/s).
  Set_vz: Set the maximum forward / backward movement speed (m/s).
  Remaing parameters are the parameters of the control algorithm.



Dip Control

Set_vx	5
Set_vz	5
Kp_v	5
Ki_v	0
Kp_dip	0.25
Ki_dip	0.1
Kd_dip	0.01

- *Horizontal Rotation Control*:
  Set_eular_y: Set the direction. This parameter does not need to be manually modified. After the UAV is started, this parameter will be automatically set as the direction of the current UAV. When the player manipulates the UAV to turn, this parameter will change accordingly with the player's input.
  Set_wy: Set the maximum rotation angular speed (° /*s*).
  Remaing parameters are the parameters of the control algorithm.
- *Height Control*:
  Set_height: Set the flight altitude. The UAV will automatically adjust to the corresponding height after startup. This parameter will also change in real time when the player inputs the up / down operation.
  Set_ Vy: Set the maximum rising and falling speed.
  Remaining parameters are the parameters of the control algorithm.



Height Control

Set_height	5
Set_vy	2
Kp_vy	0.18
Ki_vy	0.02
Kd_vy	0.05