Manuscreen user documentation

Antoine Schmoll Guillaume Grosshenny

2022-05-02

Abstract

In this paper you will find information about how to use Manuscreen and how Manuscreen works.

Contents

1 Quick start

1.1 Set up your scene

The fastest way to create a new scene in *Manuscreen* is to duplicate the given scene "SceneBase" and to rename the duplicate. In this scene you will get all indispensable elements to run a game inside *Manuscreen* already set up.

1.2 Add and manipulate objects

Inside a scene in *Manuscreen*, every object that is part of the game *(i.e. part of the level design, an object that the player can interact with, etc...)* must be placed on an *IsoGrid* and get the *Mapper* component *(the only exception is for object with the Primitive component)*.

As a quick start you can use the prefab *IsoCube1x1* located in *Manuscreen* → *IsoEngine2D* → *Resources* → *IsoEngine2D*, you can use the shortcut *CTRL+G* to instantiate a cube in your scene. This cube has the *Primitive* component which has an attribute called *Anchor*. This attribute encapsulate needed data to place an object on a grid *(the grid that the object is attached to and a position on that grid)*, you can place the object using the position components i, j and k or place directly the object on the grid using the mouse. Please take a look at the Figure 1 to see the coordinate system used by Manuscreen, be aware that i and j need to be integer and k is a float and corresponds to the height of an object.

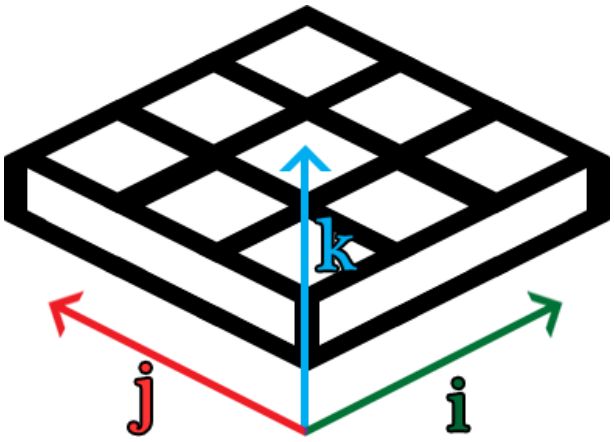

Figure 1: Coordinate system inside Manuscreen.

Using Primitives allow you to modify their size with the attributes *width*, *depth* and *height*. This is very useful to create platforms, stairs and a lot of other things while limiting the number of objects in the scene *(these attributes are in mapper component as well)*.

1.3 Set up collision

Objects using *Mapper* component are part of the collision system. You can enable/disable the collision on a *Mapper* by checking/unchecking the *Heavy* attributes and use the attribute *HeavyFilter* to control with which objects a given object can collide or not *(using name or tag)*. Keep in mind that disabling collisions will not disable the trigger collision system in Manuscreen !

1.4 Test with Gabari-San

With this plugin, you get a free customizable character named *Gabari-San*. This character has been setup to act like *MrPaf* from the game **An interesting journey of Monsieur Paf**[1] ! *Gabarin-San* has the following controls :

- **w, a, s, d** to move
- **spacebar** to jump
- **ctrl + w, a, s, d** to turn the character without moving
- **shift + w, a, s, d** to push an object while moving
- **f** to hit in front of the character
- **e** to lift the object in front of the character
  - while lifting press **e** to put the object down
  - while lifting press **f** to throw the object

2 How to set up a scene

2.1 Indispensable elements

2.1.1 IsoEngine

The *IsoEngine* component is the master of the other elements. It handles a lot of things inside *Manuscreen* and link everything together. It also tracks performance issues and how the scene should be loaded, if it should be destroyed on load, how it should be unloaded, etc... Only one object with the *IsoEngine* component has to be inside the scene hierarchy.

2.1.2 IsoScene

This component must be placed on the root of the hierarchy of the scene. It handles references of specified objects between levels, the things to do while loading/unloading a scene and can play ambient sound/music.

2.1.3 IsoGrid

The *IsoGrid* component is the grid on which every object of the game must be placed. You need at least one object with the *IsoGrid* component but you can place multiple grids in a level at the same time. You can modify the size of the grid, but keep in mind that the size of each grid impacts directly on the framerate of your game !

2.2 Best way to create a scene

The fastest and safest way to create a new scene is to duplicate *SceneBase*, this scene is already correctly set up.

[1] Steam Page of An interesting journey of Monsieur Paf

2.3 Organize your assets in your level

Each *Mapper* need to be placed inside the bounds of an *IsoGrid*, if not, it may create bugs with the rendering algorithm. Also, *Mappers* need to be children of the *IsoGrid* they belong to, if it is not the case, the corresponding *IsoGrid* will automatically be set as their parent.

3 Use Manuscreen for your project

3.1 Create your own objects

The *zOrder* defining the display order in the *IsoEngine* affects the children of the gameObject owning the *Mapper*. So you can create objects with multiple sprites. These will be included in the visual repositioning on the *IsoGrid.* However, there are a few precautions to take. It is better to put the *Mapper* component after placing all the children sprites, so as to be sure that they all get the same *zOrder*. The *Mapper* must be in the parent and **not** have a *Sprite renderer*. You can adjust the visual order of the children between them using the z position.

3.2 Add physics simulation to an object

Manuscreen offers a physics engine for simple physics interaction such as gravity, impulse and velocity. To enable physics simulation on an object, you need to add the *Movable* or *ActorManager* component. You can change gravity per object and allow an object to be pushed *(with the attribute canBePushed)* or to be lifted *(with the attribute canBeLifted)* !

3.3 Use the Interaction Manager to add interaction in your scene

One of the biggest part of *Manuscreen* is the *InteractionManager*, it provides a wide range of interaction where each one is composed of two things : a *trigger* and a list of *actions*.

- The trigger can be of different types, each one has its own attributes and behavior. The most common one is the *Event* type, which allows you to create multiple triggers such as OnTriggerEnter, OnTriggerExit, trigger the action when the object is moved, etc...
- Actions can also be of different types, that allows you to do a lot of things. The most common one is the *Callback* type, which allows you to call a function of a component of a given object at runtime.

You can find more information about the *Interaction Manager* on the *Ernestin's youtube channel* with our tutorials playlist !

3.3.1 Example : create a destructible door

First example of use of *Interaction Manager* with a destructible door *(see Figure 2)*. Start by create a primitive *(use CTRL+G)* and add an *Interaction Manager* component to the object then add an interaction *(click on the "+" button in the component)*. Once you have created an interaction, set the "*Init on start*" attribute as *true*, change the type of the trigger to *HIT* and set the *Object* attribute as *SELF (these attributes are highlighted in red in the Figure 2)*. Next you need to add an action linked to the trigger *(click on the "+" button)* and change the *type* attribute to *CALLBACK*, then add a callback, the same way you add an event in the Unity

UI system, set the door as the target and call the function Primitive.DestroyMapper *(these attributes are highlighted in blue in the Figure 2 and see the callback setup in the Figure 2).*

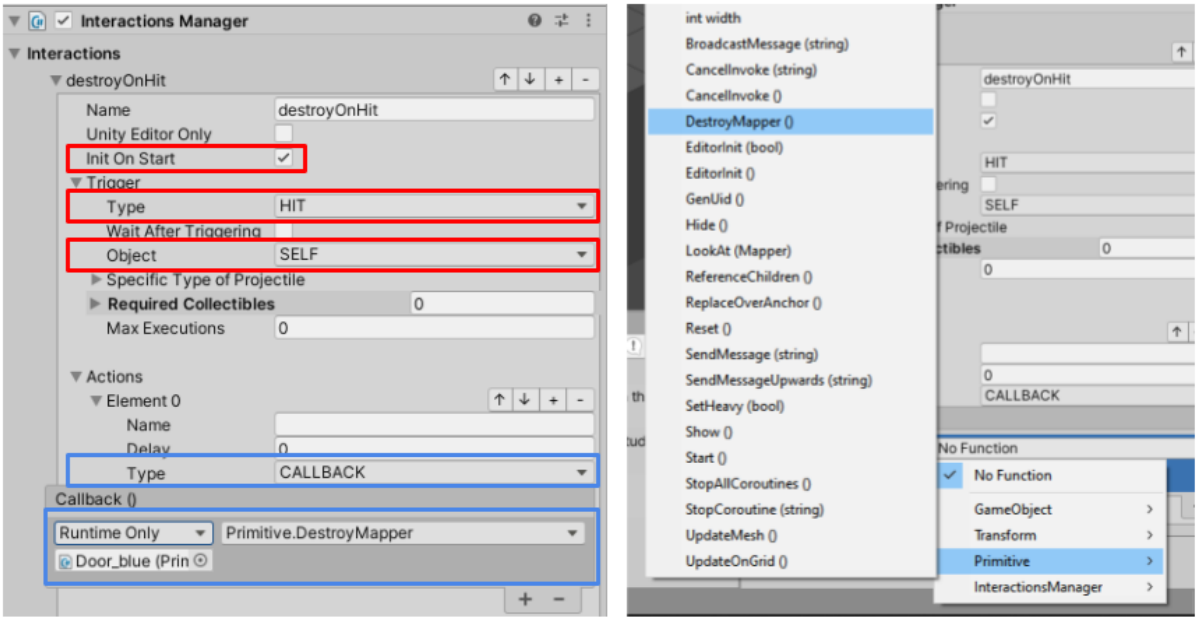

Figure 2: Interaction Manager and Callback configuration.

3.4 Add movement manager and pathfinding to object

If you want an object to be moved and to add a pathfinding behavior to an object, you need to add an *ActorManager* component. This script handles movements in all eight directions and up/down, you can use it to create your own characters *(player and NPC)*. Moreover it comes with an A* pathfinder that you can use together with the *Interaction Manager* to create basic AI !

3.5 Create a basic platform with PlatformManager

Manuscreen offers the possibility to create basic platform using the *PlatformManager* component which inherits from the *ActorManager* class. As you can see in Figure 3 the *PlatformManager* encapsulate a list of *Patterns (highlighted in purple)* where each pattern is composed of a name *(highlighted in blue)* and a list of *moves (highlighted in red)*. Each *move* consists of a movement of *one* cell in the given *direction* at given *speed*. You can also move up/down by setting the *zSpeed* value, set an *Acceleration* instead of a direct velocity and the amount of seconds to wait before doing each move with the attribute *Waiting Time Before Moving*. You can also set the reaction when something obstructs the path to handle obstacles between *Stop*, *wait*, *push* and *Next pattern*.

You can create multiple patterns in a single *PlatformManager* each one identified by a unique name. Then, you can play a given pattern either by setting the *Play on start* index as the pattern index you want to play on start or using a callback in an *Interaction Manager* with the *Play(NameOfThePattern)* method of the *PlatformManager*.

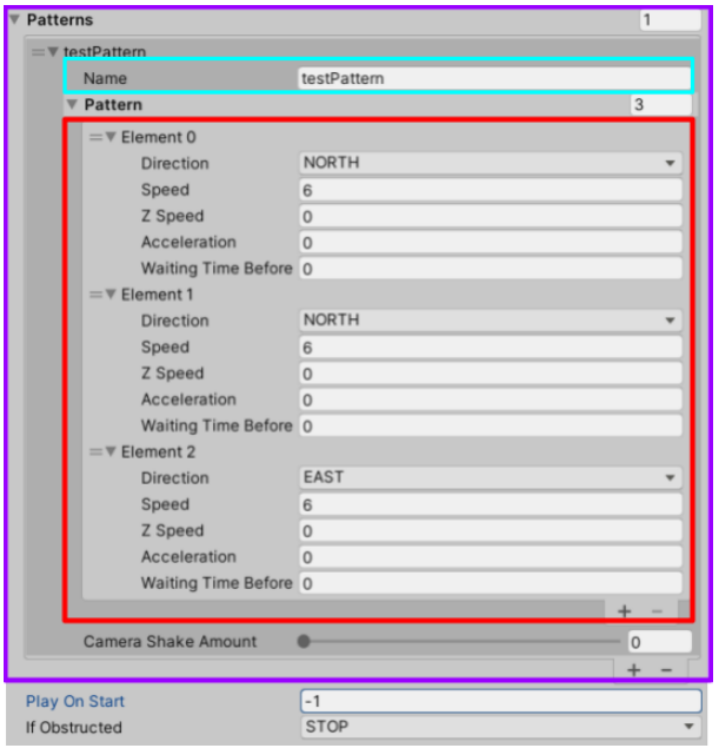

Figure 3: Platform Manager configuration.

3.6 Teleport player inside your scene and link scenes together

You can teleport the player between scenes and inside a scene. The first thing to do is to add the scenes in which the player have to be teleported inside the *build settings* in Unity, then you have to go to the folder *Resources* &rarr *IsoEngine2D* &rarr *Levels* and select the *Level* ScriptableObject. This object lists the scenes in the Build Settings, you need to check the levels in which the player can be teleported to reference it inside the Manuscreen system. Once you have done this, you need to place an object in the destination scene and set the tag *IsoReference* to create the teleporter destination point. Last, you need to add a teleporter *(CTRL+T)* and set the destination with the given scene and an object with *IsoReference* tag *(cf Figure 4)*.

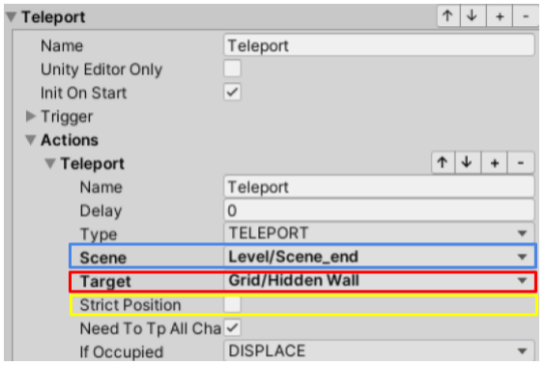

Figure 4: Teleporter configuration.

4 Main Manuscreen's scripts

4.1 IsoGrid

The *IsoGrid* class represents a grid of a given size. You can place multiple grids inside the same level and each one knows every object placed on it and handles some events such as entering or exiting a cell of the grid for instance .

4.2 Point

The *Point* structure represents a position in a given grid with i, j attributes corresponding to the cell coordinates in the grid and k corresponding of the height of an object.

4.3 Anchor

The *Anchor* structure encapsulate a *Point* object binding it to a given grid. It is possible to compare and manipulate two *Anchor* objects using the *AnchorExtension* class.

4.4 Mapper

The *Mapper* class handles the positioning of an object on a grid, manages the collision and the collision profiles. It also handles the size of an object with the given *width*, *depth* and *height*, the *width* and *depth* are integers and the height is float *(the size of an object is expressed as cell of the grid).*

4.5 Primitive

The *Primitive* class inherits from the *Mapper* class, it uses 3 meshes to create an isometric cube and scale each mesh to scale the cube to the given size. This component is really useful to quickly prototype level design.

4.6 IsoScene

The *IsoScene* class encapsulates the grids and the objects of the scene. There should be only one *IsoScene* per scene.

4.7 IsoEngine

The *IsoEngine* class centralize the other game managers to properly handle a 3D isometric scene. It has a reference to the main camera *(the one with the IsoCamera component)*, the ambient music and sfx to play during the game and more. It was made for the game "An interesting journey of Monsieur Paf", but is modular enough to be used for other projects.

4.8 MovableManager

The *MovableManager* class handles the physics simulation of an object inside the grid bounds. It also handles the movement of an object to a given direction, adding the possibility to push, lift and bind two mappers together.

4.9 ActorManager

The *ActorManager* class inherits from the *MovableManager* class. It adds an A* pathfinding, jump and other base movements to create *Playable* or *Non-Playable character*.

4.10 IsoPathfinder

The *IsoPathfinder* class implements an A* pathfinding method with three way to compute path priority :

- Natural
- Lowest_F_Score
- Lowest_H_Score

Each weight computation method changes the path that the *AI* will take to reach the destination. You can modify other data such as the path accuracy or the computation optimisation.