GDSP

GAMEDEV STARTER PACK

Manual

Jan Carlo T. Arroyo
Norman Flynn O. Dela Cerna
Lucky Sheen Prince B. Pajo
Justin Othello B. Vargas

2023

GDSP

GAMEDEV STARTER PACK

Manual

Jan Carlo T. Arroyo
Norman Flynn O. Dela Cerna
Lucky Sheen Prince B. Pajo
Justin Othello B. Vargas

Terms to Learn Before Proceeding:

Game Object = the most basic unit in Unity. It is everything from the character to the audio sources,
Prefab = a pre-made game object used for easily duplicating or replicating a single entity.
Array = a collection of multiple objects with the same object type.
Asset = art (in this case, 2D art) files, or audio files.
Parent = the game object with another game object/s inside of it.
Child = The game object inside the parent object.
Anatomy = the word anatomy is very loosely used for the purpose of using the word to describe all the parts of a template.

GAMES included in GDSP:

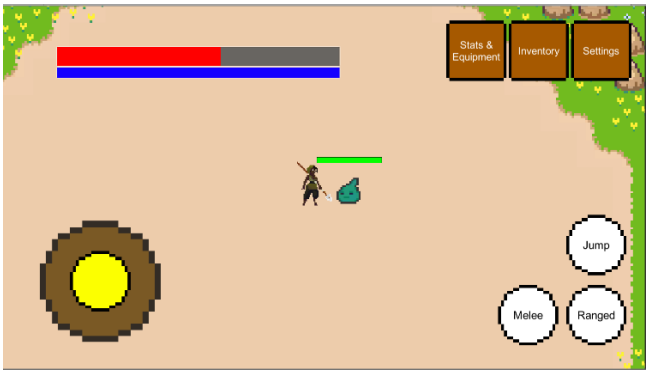
Top-down Action RPG
Platformer
Puzzle (Tile-matching, Bubble Shooter, Tetris)
Strategy (Tower Defense)

Demo Video Link to learn how to install and use GDSP:
https://youtu.be/670VGRVWPZE

TABLE OF CONTENTS

Chapter 1
Top-down RPG

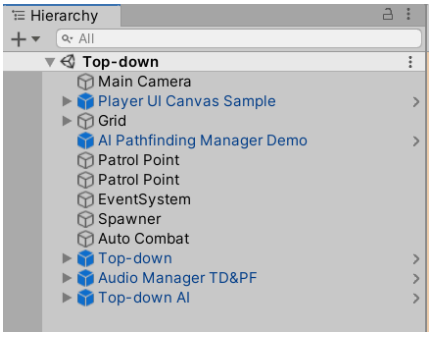
GDSP has 6 various game templates. These templates are modeled from some of the most famous games across generations. While there are only six, this is already plenty to start making game prototypes.



The GDSP Top-down RPG Template on Play in Unity

The **Top-down RPG** template of the GDSP is modeled around beat 'em up and hack and slash games, with a bit of level progression and item usage for health and skill. The template can be used for multiple platforms, but since GDSP lacks the online capabilities, it would not allow cross-platform, unless you, the end user, add online functionalities.

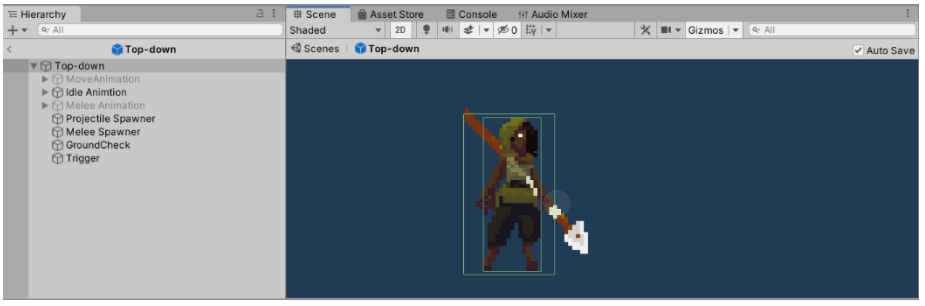
Using the Top-down RPG template allows you to create top-down games instantly. If you want to just test your assets for your game, the template can also be used to do so. Here is the anatomy of the template:



The anatomy of the Top-down RPG Template. The blue ones are prefabs which are basically templates within templates and can be used for other purposes.

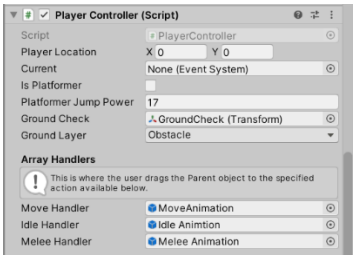
- Main Camera
- Player UI Canvas Sample
- Grid
- AI Pathfinding Manager Demo
- Some Patrol Points for the AI
- EventSystem
- Spawner
- Auto Combat
- Top-down
- Audio Manager
- Top-down AI

Player Character (Top-down)



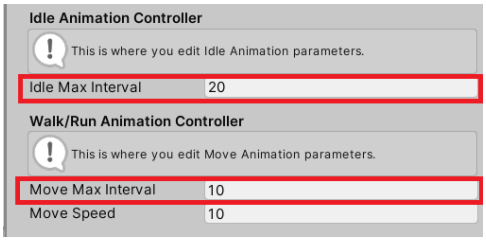
The Anatomy of the Player Character

First to discuss is the highlight of the template: the player character named Top-down, which is third from the bottom. Modifying the character is easy. Just look at the inspector and find the Player Controller script.



A portion of the Player Controller Script in the Inspector

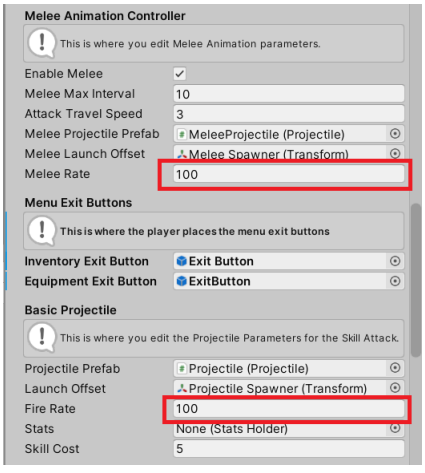
The easiest way to modify the character is to adjust the animation speed of the character. We can see that there are three types of animation controllers, namely: Idle, Walk/Run, and Melee. As we can see on the package, there are variables with "Max Intervals" on their names. Those variables control how fast the frames of your animation changes. The higher the number, the slower the frames change.



Sample of the Max Intervals

Next would be the movement speed of the character. This is how fast or slow the character moves. The GDSP Character template does not include a sprint/dash action, so keep that in mind when setting the character speed. To change the speed, similarly to what is stated above, you can change the character speed by changing the values of variables with "Speed" such as Move Speed and Attack Travel Speed. The higher the number, the faster the character moves (*see **Move Speed** from the previous image for reference*).

Another easy-to-edit variables are the attack speed for both melee and ranged. To change the melee attack speed, see the Melee Animation Controller Section of the Player Controller Script, and change the value of Melee rate to your liking. To change the range projectile attack speed, adjust the Fire Rate. The Skill Cost variable is linked to the stats of the player (which will be discussed later). If you want unlimited ranged attacks, just set it to zero. Like animation controllers, the higher the number, the slower the attack.

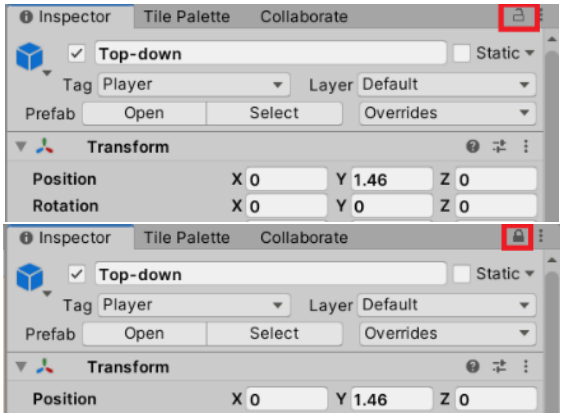


The attack speed variables of the player controller.

Now comes the more complex parts. We'll start off with the UI controls. Since the script holds the player controls, the UI controls are also bound to the script. Now, let's assign the components needed for the UI. But before that, we need a quick tutorial!

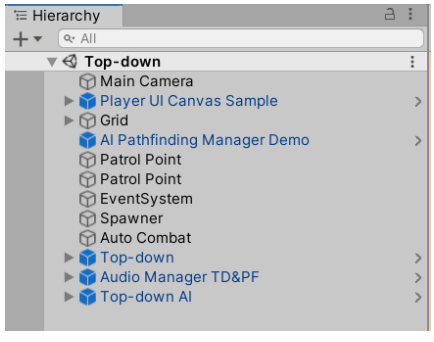
A Quick Tutorial into How to Drag-and-Drop Inside Unity

Note: To make the drag-and drop less annoying, click the Lock icon on the top right of the inspector shown below. Upon clicking, it should turn into a darker color with the lock icon being locked. Make sure to unlock after assigning the needed components.



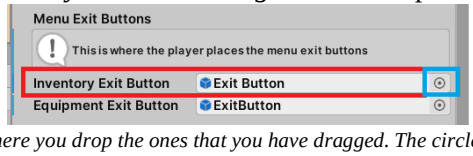
The before and after of clicking the Lock icon.

Step 1: Select the Game Object you want to change components to in the Hierarchy.



Step 2: Lock (see steps above).

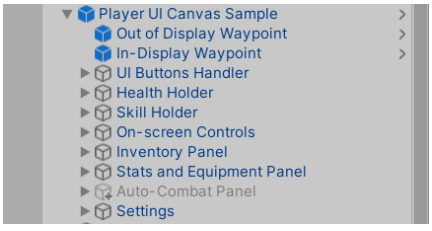
Step 3: Find the component you want to assign a new component to.



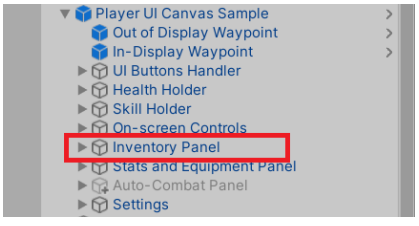
The component in red is where you drop the ones that you have dragged. The circle with the dot in the middle indicates that you can assign another object on the slot instead of a number of a word.

Step 4: Find the component that you want to assign on the slot. Since this example needs the Inventory Exit Button, let's locate it.

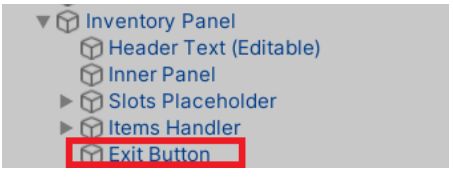
- *Click the Arrow down to expand the Template. Do not double click or click the arrow facing right because it opens the prefab. We don't want that to happen when assigning components.*



- *Since we are looking for the Inventory Exit Button, let's expand the Inventory Panel.*



- *We can see an object named Exit Button. Now that we have found the component we need, lets drag and drop it to the Inventory Exit Button slot.*



Now that we know how to assign components to where they are needed to assign, the manual will now list the locations as this:

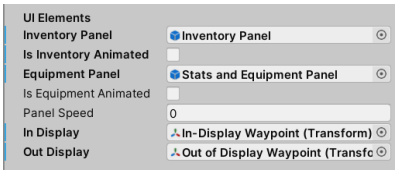
- *Main Parent*
  - *Child 1*
    - *Child of Child 1*
      - *Child of Child of Child 1*
        - *And so on…*

For Example:

- *Player UI Canvas Sample*
  - *Inventory Panel*
    - *Exit Button*

After knowing the basic steps in dragging and dropping, you now learned the most important part of GDSP.

End of the Quick Tutorial into How to Drag-and-Drop Inside Unity



The UI Elements Section

Now that we know how to assign the Inventory Exit Button, we can now also assign the Equipment Exit Button. But before that, the reason we need to assign these is for the other platforms outside of touch and mouse/keyboard. The script would select those two as priority for the selected button every time it is opened. We can find the Equipment Exit Button from this location:

- *Player UI Canvas Sample*
    - *Stats and Equipment Panel*
        - *ExitButton*

Let's skip to the UI Elements section. We need to assign the Inventory Panel, along with the Equipment Panel from the Player UI Canvas Sample. These are their locations:

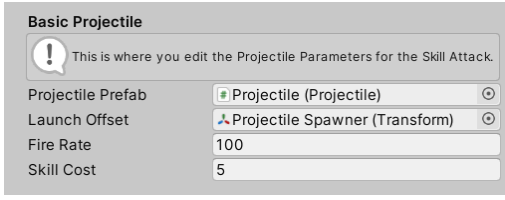
- *Player UI Canvas Sample*
    - *Inventory Panel (for Inventory Panel)*
- *Player UI Canvas Sample*
    - *Stats and Equipment Panel (for Equipment Panel)*

Next would be the In Display and Out Display. These are the locations as to where the UI panels are placed when used and not used. The In Display is the position for when it is shown and vice versa for the Out Display. The purpose of these objects will be further discussed on the UI part for the Top-down RPG template. Now, let's assign it by getting the components from these locations:

- *Player UI Canvas Sample*
    - *Out of Display Waypoint (for Out Display)*
- *Player UI Canvas Sample*
    - *In-Display Waypoint (for In Display)*

The items with the checkboxes are toggles. In this case, the Is Inventory Animated and Is Equipment Animated have toggles that we can check and uncheck. If we check the boxes, it sets the function that the panels will be animated. If animated, the panel will move from the Out Display to the position of the In Display if opened, and vice versa when closed. The Panel Speed is how fast the panel will travel. The higher the number, the faster it moves. If the boxes are unchecked, the panels will instantly pop-up from the display locations.

The last section that we can customize in the script is the Basic Projectile section. This sets the projectile the player releases, the location of where it shoots, the speed of shooting, and the cost.



The Basic Projectile Section

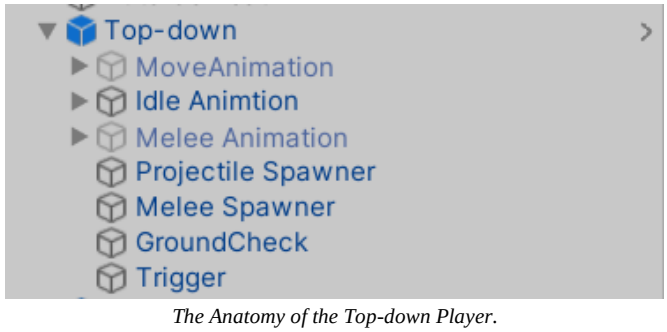
The projectile prefab slot will hold what projectile the player will throw. We can find other projectiles from the Templates folder in the Assets Window. The prefabs are found in this location (*disregard the Scenes folder*):

- *GameDev Starter Pack*
    - *Templates (this is where the Projectile Prefabs are found)*

The Assets Window

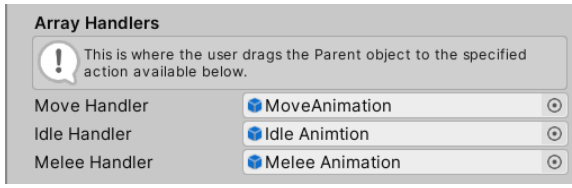
Now is the launch offset. This is where the projectile is spawned. To locate the position of the launch offset, here is the location in the Hierarchy:

- *Top-down*
    - *Projectile Spawner*



The Anatomy of the Top-down Player.

The Fire Rate and Skill Cost were already discussed earlier, so we can skip to those parts.



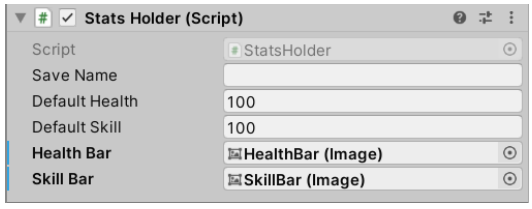
The Array Handlers Section

While not important in this context, the Array Handler section contains the parents of the sprites (which will be discussed at the Customization chapter).

To set the stats of the player, the Stats Holder script is responsible for doing the job. This is where you set the default values for the player, and the images that goes up or down when the player takes damage or use a skill, and when the player replenishes its health and skill (basically works like mana). Just don't set the default health to zero, and don't set the default skill to zero if you have added a value on the skill cost from the Player Controller. The Save Name is optional, but you can assign something, and it will not cause any error. These are the locations of the images in Hierarchy:

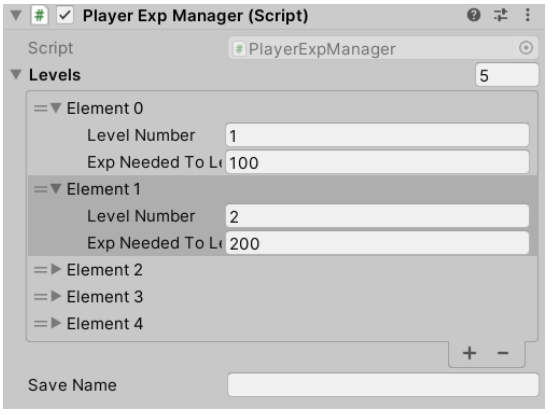
- *Player UI Canvas Sample*
    - *Health Holder*
        - *HealthBar*

- *Player UI Canvas Sample*
    - *Skill Holder*
        - *SkillBar*



The Stats Holder Script

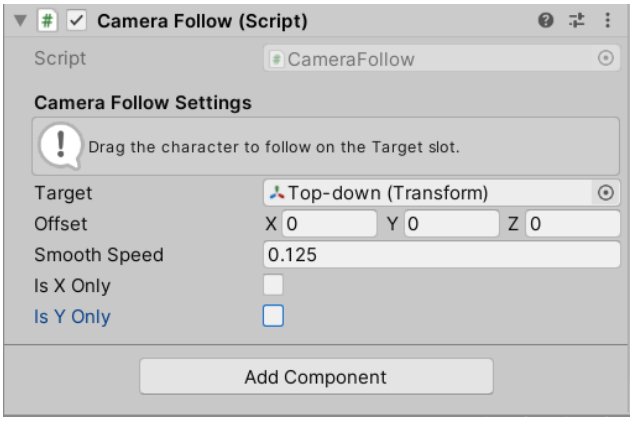
To assign the levels, the Player Exp Manager is your go-to script. The array of levels is where you can add the levels for the player. You can expand the levels and further expand the elements to see the contents. To add a level, just click the "+" icon below the levels section, and set the level number to the next one, and set the exp needed to level up to your preference. The tip is that the higher the level number, the greater the exp needed to level up. The Save Name is optional, but you can assign something, and it will not cause any error.



The Player Exp Manager Script with the expanded Levels

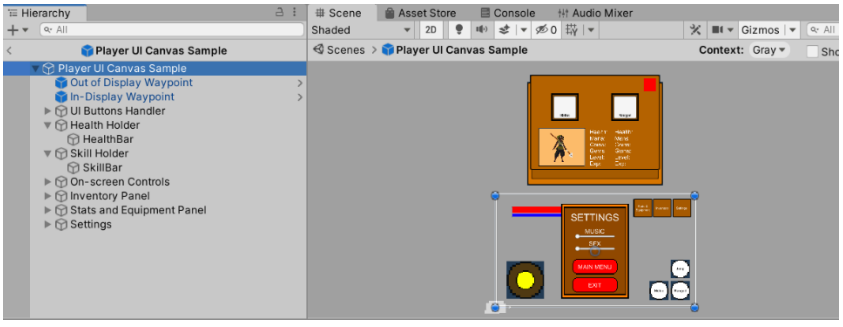
Main Camera (Top-down & Platformer)

While the Main Camera has other scripts attached to it already, usually, those are Unity's default values. The one that we can customize is the Camera Follow script. For the Target, just drag and drop the player character in that slot. The use of offset is optional but if you want the camera to be offset from the character, you can do so. The X value is for the horizontal, while the Y is for vertical. While the Z is shown, it is best that it is set to zero. The smooth speed is how fast the camera follows the player character. The value 0.125 is the sweet spot but you can still experiment the smooth speed. The higher the number, the faster the camera follows. The Is X and Is Y Only toggles are for setting limitations on the direction the camera can follow. If Is X Only is checked, the camera will only follow on the horizontal direction. For the Is Y Only, the camera will only follow in the vertical direction. If you want the full range of motion, just uncheck both.



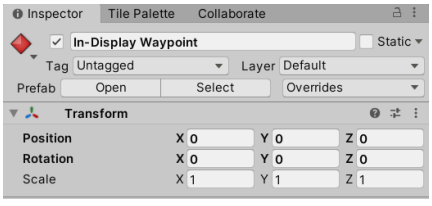
The Camera Follow Script attached to the camera.

UI (Player UI Canvas Demo, Top-down & Platformer)



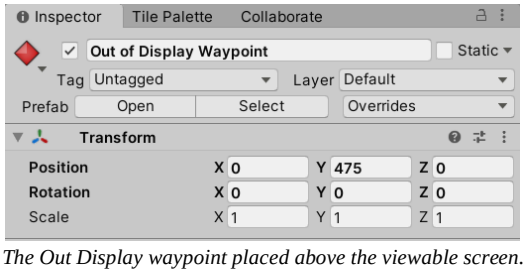
The Anatomy of the UI Template

Let's start with what we have touched before, the In and Out Displays. The purpose of the In Display is to position the animated panel dead center of the screen, and to do so, just set its position to (0,0,0) like this.



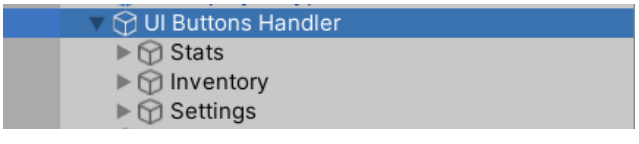
The In Display waypoint which the position was set to (0,0,0)

Vice versa, set the position of the Out Display anywhere outside the viewable area. Keep in mind that the direction of where you put your Out Display is the determining factor of where the panels will come from. If you placed the out display on the far left, the UI panels will also come from the far-left side.



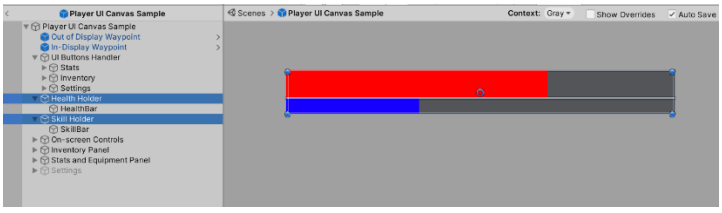
The Out Display waypoint placed above the viewable screen.

The UI Buttons Handler child just holds the Stats and Equipment, Inventory, and Settings buttons. If you have no prior experience in using Unity, just don't touch this area.



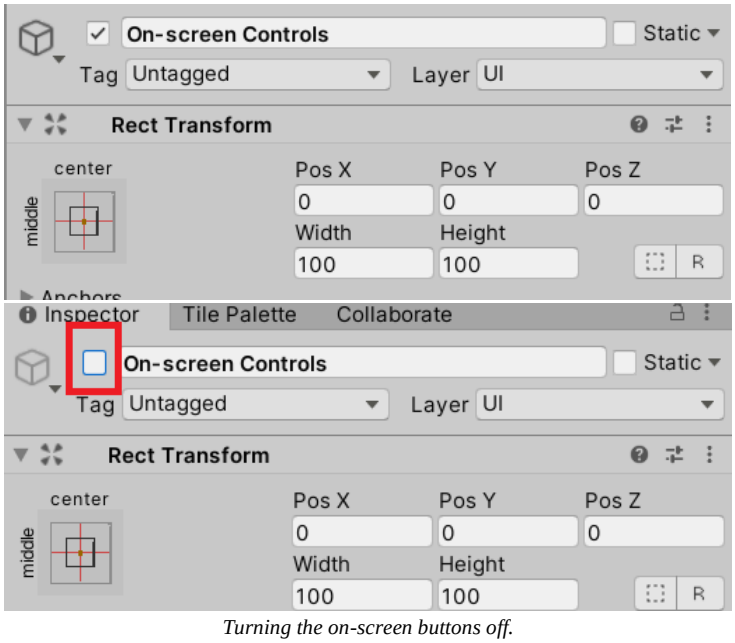
The UI Buttons Handler and the button children

Next would be the Health and Skill Holder images. These images serve as background for the indicators for the health and skill discussed previously.

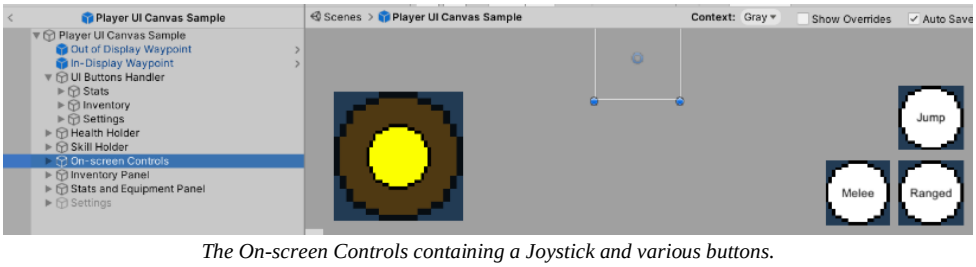


The Health and Skill Bars

The On-Screen Controls child contains the buttons used for touch. If you don't want to have a hovering button on your game, just turn off this child like this:

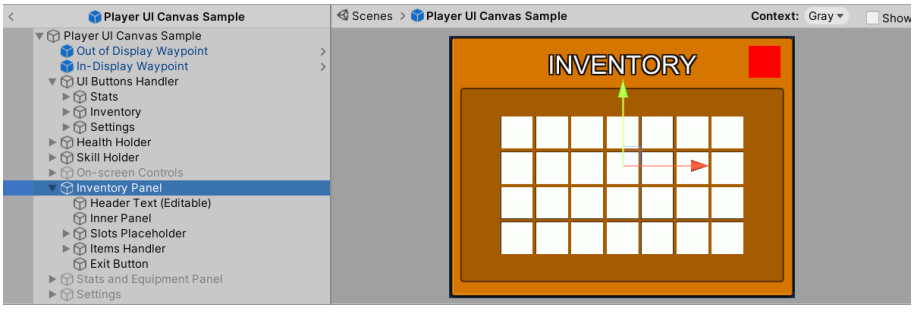


Turning the on-screen buttons off.



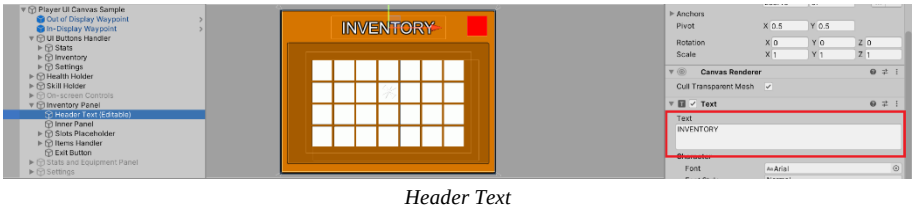
The On-screen Controls containing a Joystick and various buttons.

The Inventory Panel handles all inventory functionalities, from getting the items, using the items, and saving it for later.



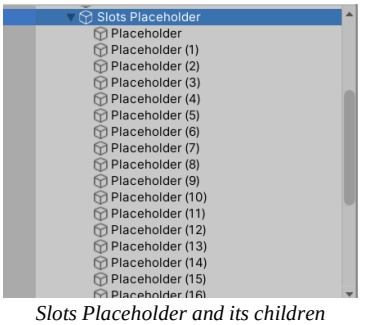
The Anatomy of the Inventory Panel

The Header Text just sets the header text for the inventory panel. We can see that the Text component in red has the word "INVENTORY". To change the header, maybe you want it to be called bag instead, just change the text to "BAG" or anything.



Header Text

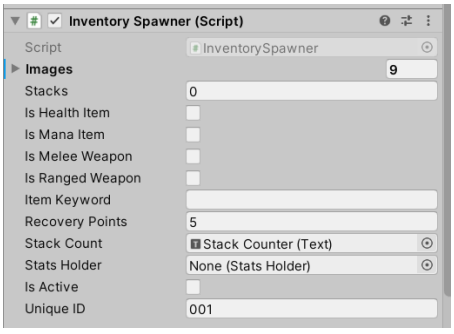
The Slots Placeholder is basically a grid which is populated by the inventory slots. If the slot is empty, this is the part that you can see. You can adjust the size of it by adding the placeholder objects by duplication (CTRL + D) or deletion by just basically deleting some placeholder game objects.



Slots Placeholder and its children

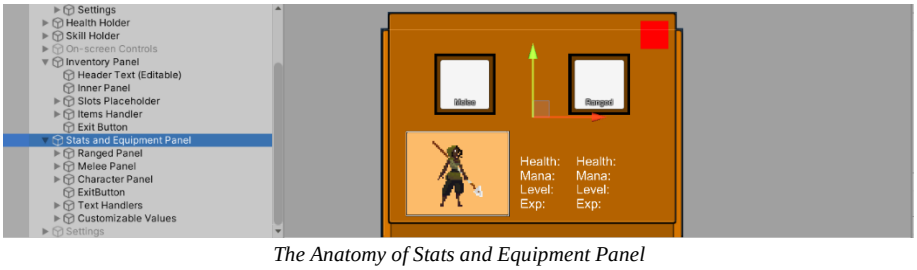
Next is the Items Handler. Like the Slots Placeholder, it is also a grid full of inventory slots, but instead of it being a placeholder, this contains the actual inventory items. You can adjust the size of it by adding the items objects by duplication (CTRL + D) or deletion by just basically deleting some placeholder game objects. Just make sure that it is the same amount as the placeholders so that it will not look awkward. Most importantly, every child of the Items Handler must have a unique identification which is located at the bottom part of the Inventory Spawner script. The Unique ID is

the unique identifier for the save system, so if you have an ID similar to another, it would not save.



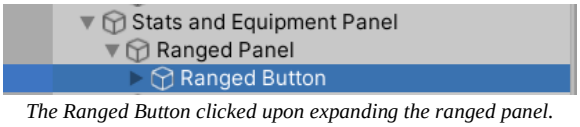
The Inventory Spawner Script for every inventory entry

Next is the Stats and Equipment Panel. This shows the player's stats such as health, mana(skill), level, and experience. It also shows the animated character idling, and the current items equipped.



The Anatomy of Stats and Equipment Panel

The Ranged and Melee Panel holds what the player is currently equipped with. If you want to add new weapons for the panel to show, expand either Ranged or Melee Panel and click the button inside.



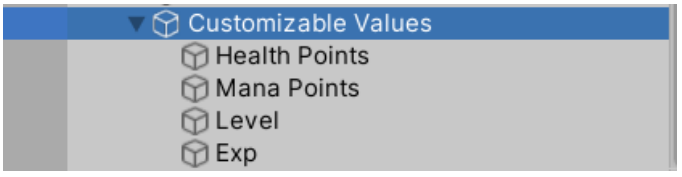
The Ranged Button clicked upon expanding the ranged panel.

You can then see the Image Handler For Equipment script. Just expand the Reference Image, and you can now drag and drop the reference image for your items. For this to work, the name of the item and the reference image must be the same. You can source your very own reference image and expand the contents of your game.

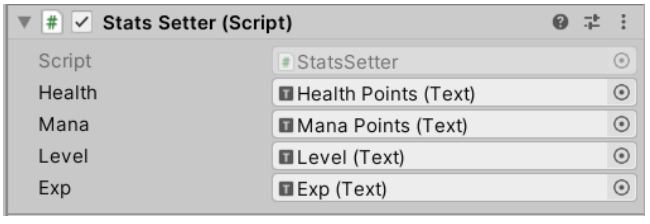
The Image Handler for Equipment Script

The Character Panel, Exit Button, and Text Handlers are not required to be changed, but you can modify these parts if you want. The one you will need to be

mindful of is the Customizable Values, as this holds the changing values that you see when looking at the Stats and Equipment Panel. You can then plug these objects to the Stat Setter Script that is Found on the Stats And Equipment Panel.

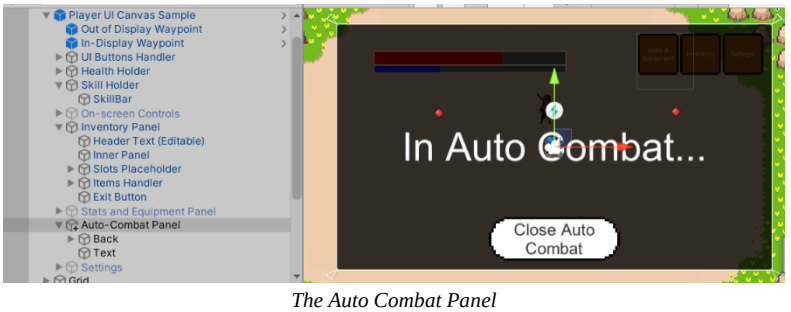


The Anatomy of The Customizable Values



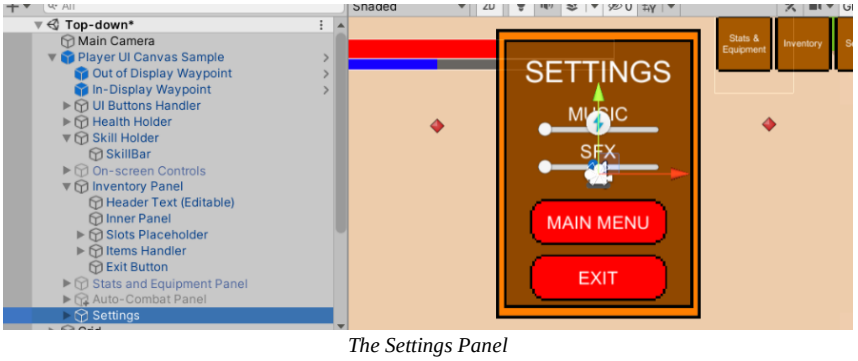
The Stats Setter after being assigned by the objects.

Next is the Auto Combat Panel. This panel shows when the player toggles the auto combat mode. The function of the auto combat will be discussed later in the manual.



The Auto Combat Panel

Last for the UI is the settings. It just contains the sliders for the music and SFX, a button prepared for going to main menu, and an exit button.

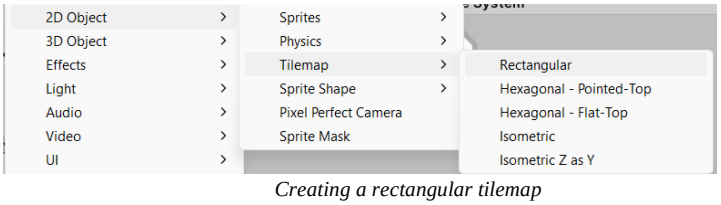


The Settings Panel

Grid (Strategy, Top-down & Platformer)

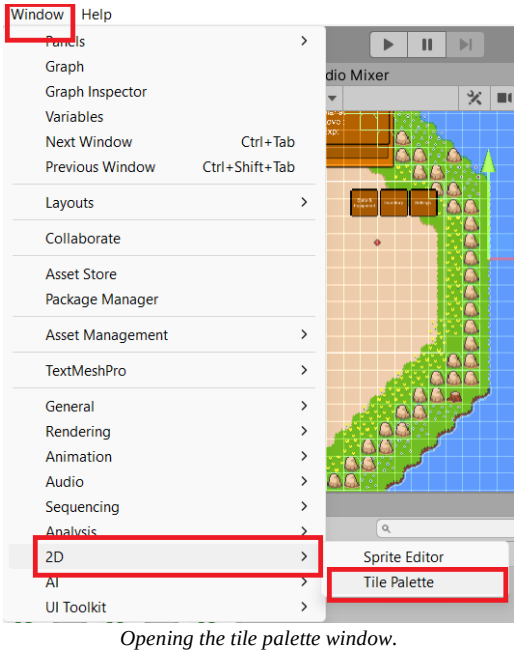
The Grid and its children

The Grid is the one responsible for making the background environment for the game. This is created using tilemaps. To create your tilemap, right click on the hierarchy, and create new 2D object, then click Tilemap, then Rectangular. If you have isometric or hexagonal designs for your tilemap, you can use those instead. For tile palette creation, refer to the Customization chapter.



Creating a rectangular tilemap

If you have a tilemap set up already (GDSP also has premade tilemaps), you can paint the scene with the design of your choice. If your tile palette is not shown, you can open the Tile Palette window by going to Window, then 2D, then Tile Palette.

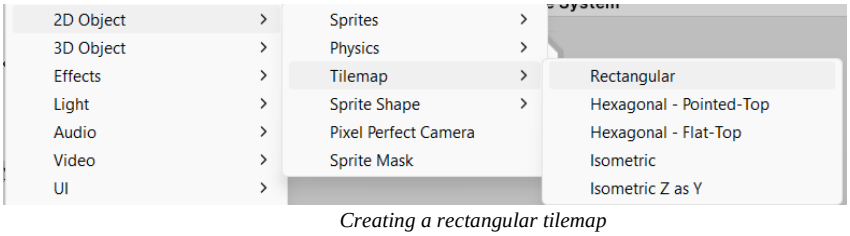


Opening the tile palette window.

Now, you can view the tile palettes the GDSP has provided or your own if you managed to create one after reading the Customization Chapter.

The Tile Palette

The Grid is the one responsible for making the background environment for the game. This is created using tilemaps. To create your tilemap, right click on the hierarchy, and create new 2D object, then click Tilemap, then Rectangular. If you have isometric or hexagonal designs for your tilemap, you can use those instead. For tile palette creation, refer to the Customization chapter.

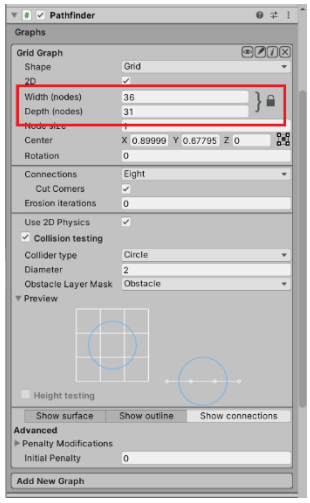


Creating a rectangular tilemap

AI Manager (Top-down & Platformer)

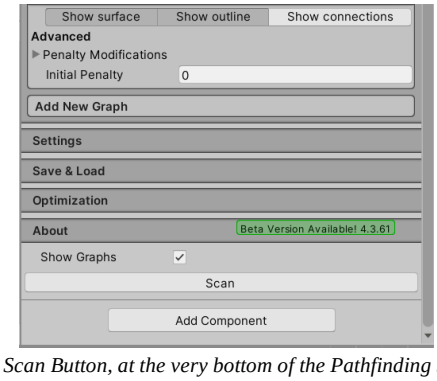
The AI Manager holds the grid where the AI can do the pathfinding function. The AI would not go out of the grid creating by the Pathfinding function. You can use this to your advantage. You can place multiple AI managers on map while assigning select few opponents that would only be on that area. To adjust the size of the grid, just adjust the width and depth. For the obstacles, GDSP has already provided an

Obstacle Layer Mask for placing the obstacles. For adding obstacles, see the Customization Chapter.



The Pathfinder Script of the AI Manager

If you have finalized the size, you can now scan the grid for obstacles so that the AI would navigate around those.



The Scan Button, at the very bottom of the Pathfinding Script

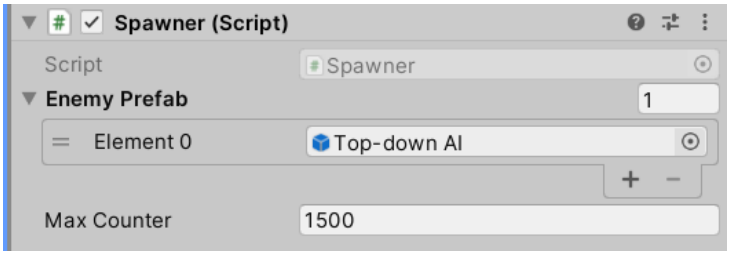
Patrol Points and EventSystem (Top-down & Platformer)

The Patrol Points are specified points in the map where the AI can move to if it cannot detect the character. If you want to change its position, you can do so by. Changing the Position of its Transform component.

The EventSystem is a +1 to the UI canvas. Its function is to handle events that are used by the UI.

The Spawner is specified point in the map where the enemy spawns. If you want to change its position, you can do so by. Changing the Position of its Transform component. You can also add variety to the spawned enemies by dragging and

dropping another AI enemy template to the Enemy Prefab array. The Max Counter is how fast the spawner spawns the enemies. The higher the number, the slower it spawns.



The Spawner Script

Auto Combat (Top-down)

The Auto Combat game object contains the function Auto Combat for Top Down, which controls the state of the game when auto combat is on (default binding is Left Tab). The Auto Combat Panel is located at:

- Player UI Canvas Sample
    - Auto-Combat Panel

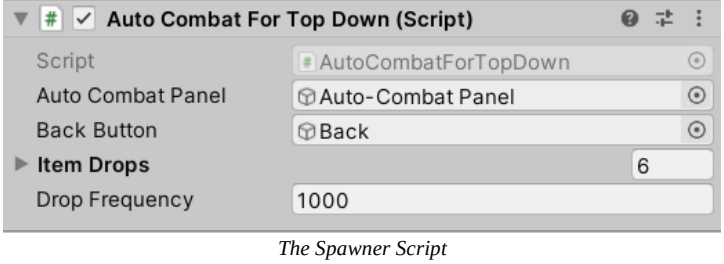
The back button is located at:

- Player UI Canvas Sample
    - Auto-Combat Panel
        - Back

The item drops are the items which will drop by doing auto combat. You can add more items by creating new item variants from the following location on the Asset Window:

- GameDev Starter Pack
    - Templates
        - TD&PF
            - Top Down

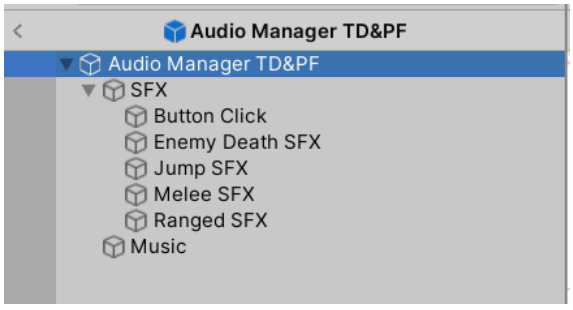
…as this folder contains the prefabs for the items for the top-down template.



The Spawner Script

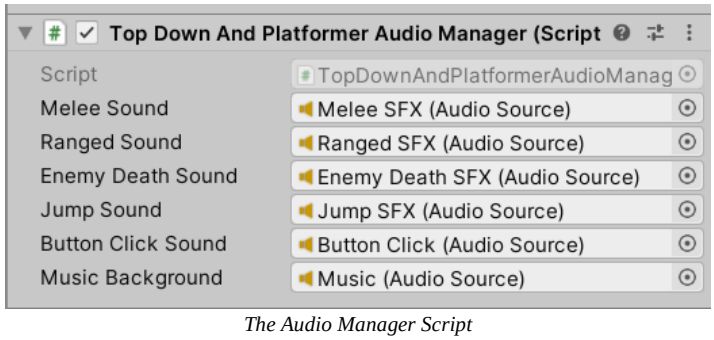
Audio Manager (All Templates)

The Audio Manager, as the name suggests, manages the audio on a scene. Its children are SFX and Music, respectively.



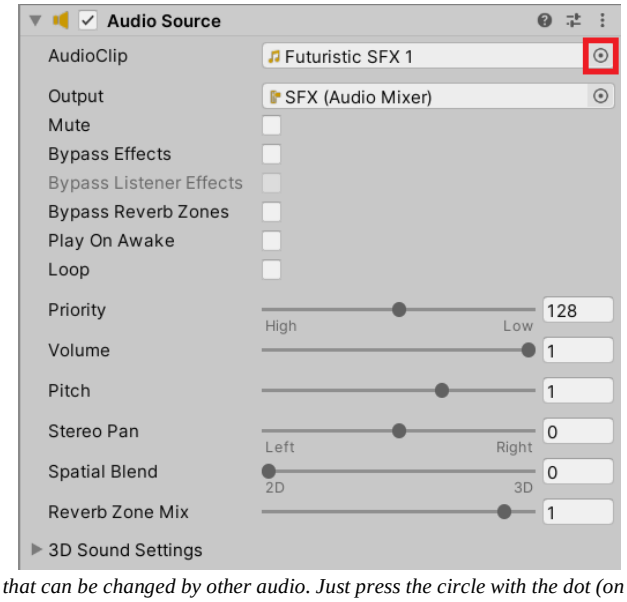
The Anatomy of an Audio Manager

With these, you can assign specific audio for specific functions such as melee SFX, ranged SFX, and more.



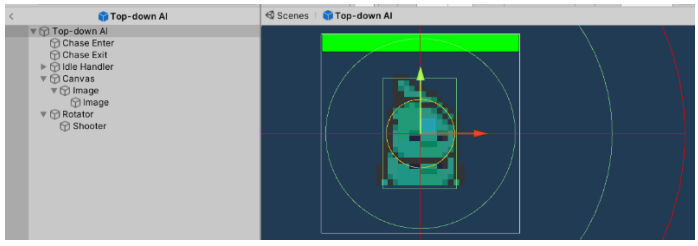
The Audio Manager Script

You can change the audio it produces by clicking on a child of the Audio Manager and changing the Audio Clip.



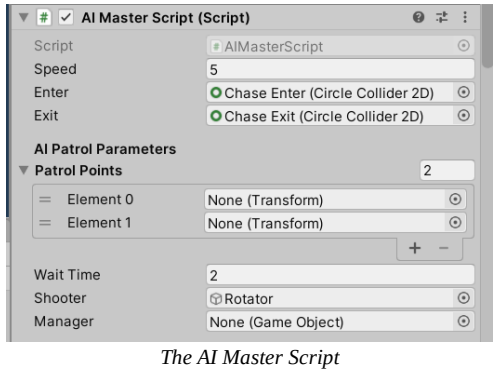
The Audio Clip slot that can be changed by other audio. Just press the circle with the dot (on red) to explore other audio files.

AI Enemy (Top-down, Platformer, and Strategy)

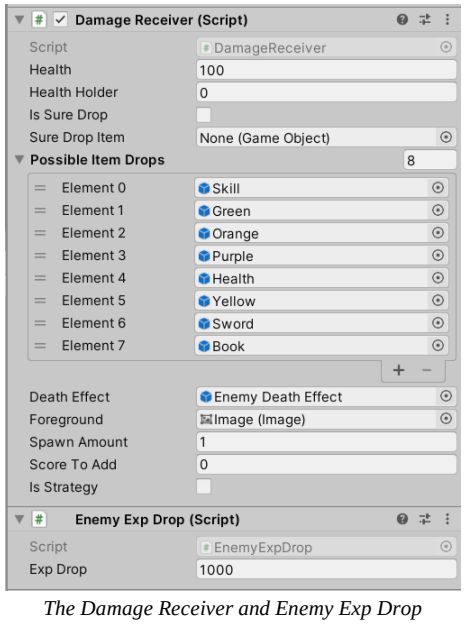


The Anatomy of an AI Enemy

The last one for this template is the AI enemy. Upon clicking, you might be overwhelmed by how many scripts attached to it. Don't be afraid since there are only three that you need: The AI Master Script, the Damage Receiver, and the Enemy Exp Drop.



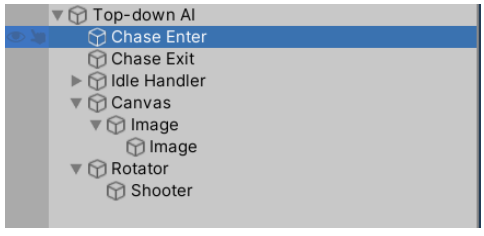
The AI Master Script



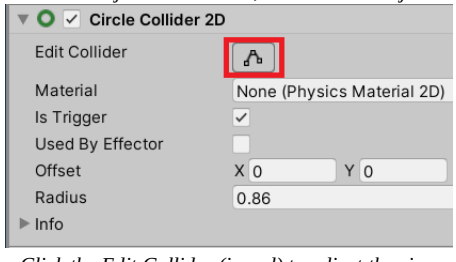
The Damage Receiver and Enemy Exp Drop

Let's look at the AI Master Script first. The easiest thing we can adjust from here is the Speed. This is how fast or slow the enemy AI patrols depending on your input. The higher the number, the faster the patrol movement.

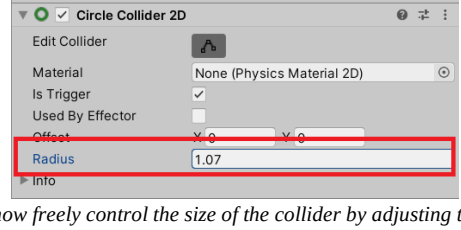
Next are the Enter and Exit Colliders. We can locate these colliders as children of the AI. The Enter collider activates the auto follow on the player. The Exit collider turns off the auto follow. It is recommended to have a larger Exit collider rather than the Enter collider so that you need to go closer to the enemy for you to be detected, but the enemy can chase you on a much larger distance. You can change the sizes of these colliders like this:



Select a collider to adjust. In this case, the one to be adjusted is Enter.

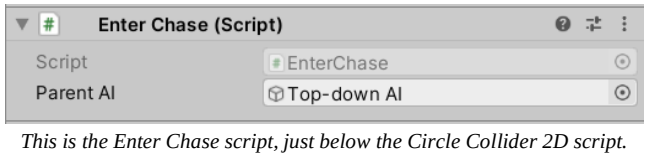


Click the Edit Collider (in red) to adjust the size.



You can now freely control the size of the collider by adjusting the Radius.

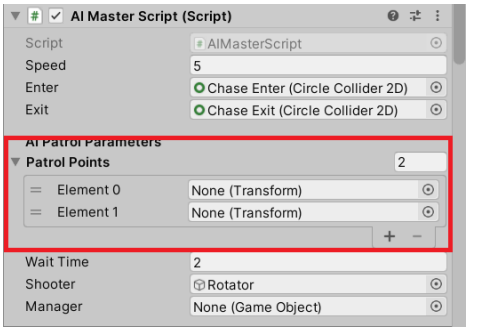
Additionally, while it is pre-assigned in the template, you can still assign manually the parent game object for the Chase colliders. Just drag and drop the AI parent to the slots needed to be filled in.



This is the Enter Chase script, just below the Circle Collider 2D script.

Next would be the Patrol Points. Since it was already discussed earlier, let's just skip to the function. Just drag and drop the Patrol Points to the indicated slot shown below. If you want to add a new set of points, clear it first by changing the size to zero

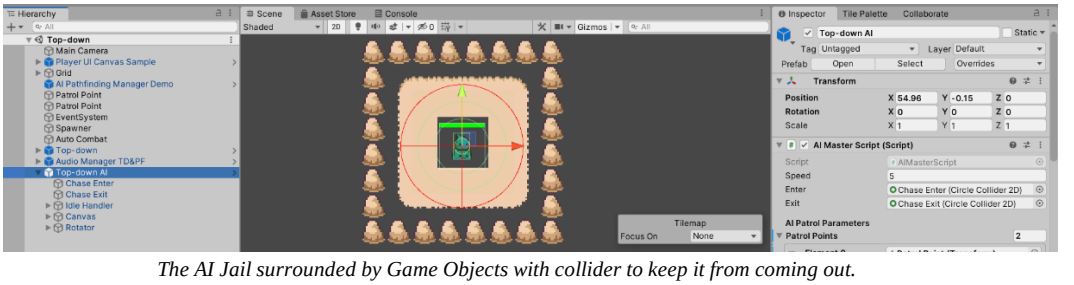
then adding new points into the slot in order. If you just want to add new points, you can just easily drag and drop. Don't forget to lock!



This is the Enter Chase script, just below the Circle Collider 2D script.

The Wait Time is how long the AI would stay in the patrol point before moving to the next. The shooter is basically a tower defense tower attached to the AI. This will be skipped for now, but it will be highlighted in the Customization Chapter. The Idle Handler will also be discussed at the Customization Chapter.

For the AI with patrol points to work when using the spawner. You need to create a reference AI for it. Assign the patrol points beforehand and place the AI on a really secluded area that is also entrapped with colliders so it would not sneak out. If you happen to kill this reference AI in game, all AI that uses that as reference would lose all their AI functionalities. On this template, we have included the AI jail located at the far right side of the scene. You can now then add the enemy AI to the spawner.

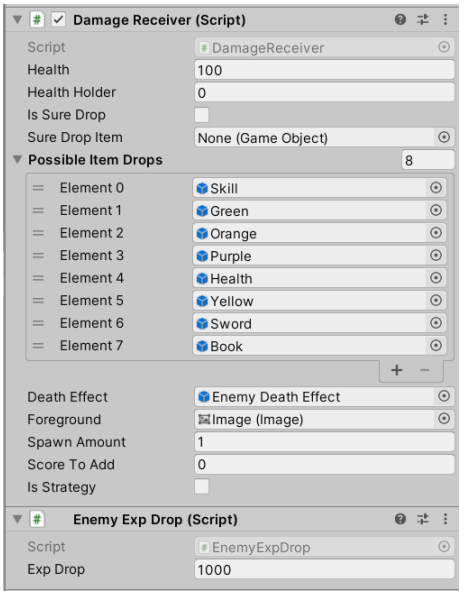


The AI Jail surrounded by Game Objects with collider to keep it from coming out.

Now, let's go to the Damage Receiver and Enemy Exp Drop. For the Damage Receiver, you can set the health of the enemy on the Health component. The health Holder just holds the current health. You can simply just ignore it. The Is Sure Drop toggle is a checkbox where if checked, would 100% drop the item assigned in the Sure Drop Item. Assigning drop items for the enemy AI is like assigning drop items to the Auto Combat manager discussed previously, so feel free to use it as reference. Same goes to Possible Item Drops. You can also change the Death Effect, but it will be discussed in the Customization chapter.

The Spawn Amount is how much items it will spawn upon death. Score to Add is how much score is added to the player upon enemy death. The Is Strategy checkbox

is for choosing the game mode. If the game template is strategy, the Score to Add will be used. If not, the Spawn Amount is used.



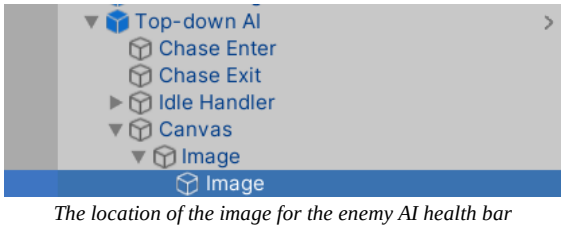
The Damage Receiver and Enemy Exp Drop

The next would be the Enemy Exp Drop. You can set how much the player can gain after defeating the enemy type.

The last to be discussed for AI is the Canvas. This holds the health bar of the enemy AI. Same as assigning the health and skill bar for the player character, you can find the health bar of the enemy AI by going to this location by using the Top-down AI as example:

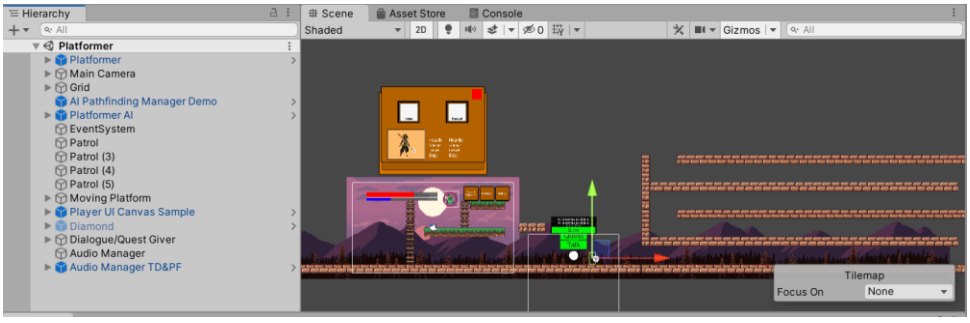
- Top-down AI
    - Canvas
        - Image
            - Image

…and you can now then assign the Image to the Foreground component in the Damage Receiver.



The location of the image for the enemy AI health bar

Chapter 2
Platformer



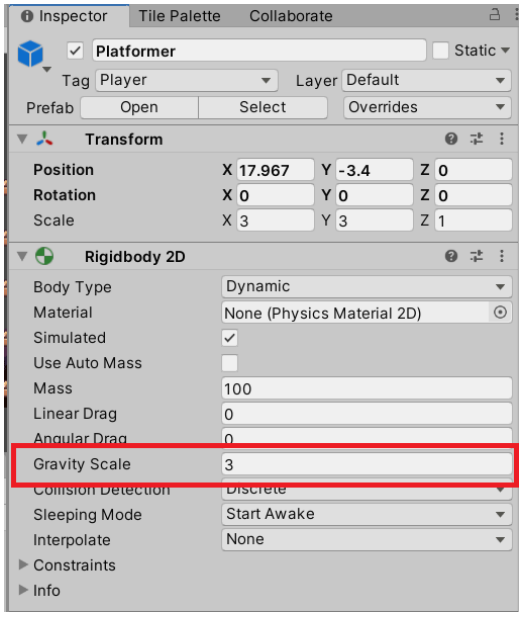
The Anatomy of the Platformer Template

The **Platformer** template of the GDSP works similarly to the previous template, but the difference between the two is that this template simulates gravity, which is what makes the character jump to and fall from the platforms, hence, it is named platformer. The closest games that you can compare this template to are Super Mario Bros., Celeste, and more. Here's the anatomy of the Platformer Template:

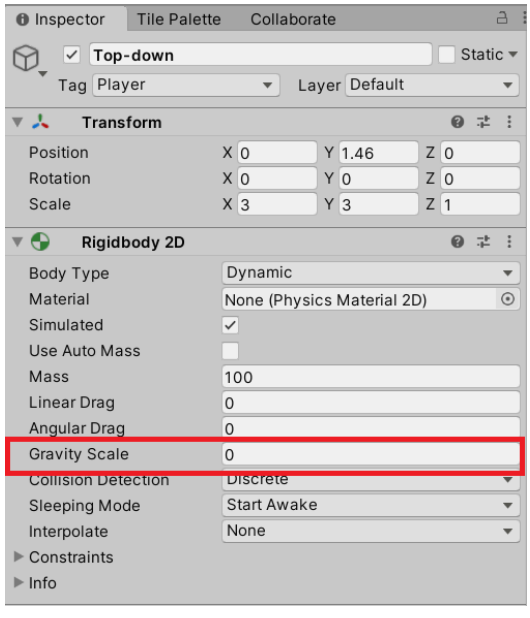
- Player Character (parts different from the Top-down)
- Main Camera (already discussed in the previous chapter)
- Grid (already discussed in the previous chapter)
- AI Manager (already discussed in the previous chapter)
- Enemy AI (already discussed in the previous chapter)
- EventSystem (already discussed in the previous chapter)
- Patrol Points (for AI and Moving Platforms)
- Moving Platforms
- UI (already discussed in the previous chapter)
- Dialogue/Quest Giver
- Audio Manager (extended)

Player Character (Platformer, extended)

Now, we shall discuss the differences between the Top-down Character and the Platformer Character. One major difference is that the Platformer character uses the Gravity Scale of the Rigidbody 2D component while the Top-down does not. Note that you should manually adjust the gravity scale depending on the use of the character.

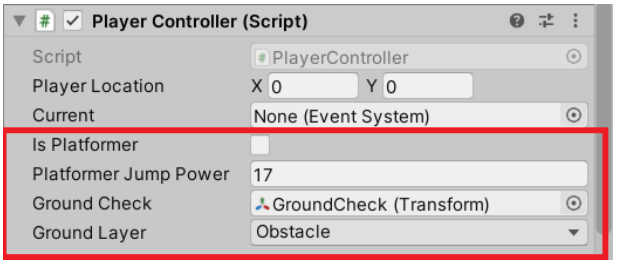


The Rigidbody 2D of the Platformer. The Gravity Scale stands for how much the constant of gravity is multiplied. In this case, 3 times, which means that the simulated gravity is more or less 3 times the gravity we experience.



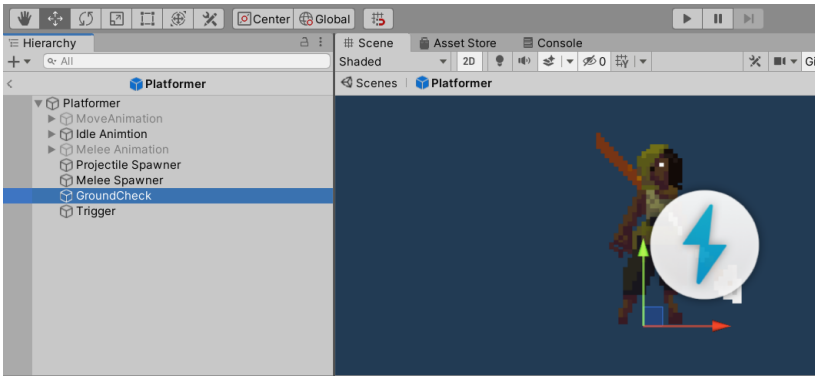
The Rigidbody 2D of the Top-down. Zero on the gravity scale means that gravity will not be simulated.

Next is the editable parts reserved for the Platformer Character. The Is Platformer checkbox should be checked if the usage of the character is for Platformer. If not, it should default into a Top-down character. The Platformer Jump Power is how high the character can jump. The higher the number, the higher the character can jump. A good tip for setting the value is adjusting the jumping power for the biggest jump that you have in the game. That should be your jumping power.



The editable components reserved for the platformer.

Next is the character controller is the Ground Check. Its purpose is to check if the player is stepping on some sort of ground, specifically the Obstacle Layer. It should be placed as equal as the foot level of the character to ensure that it can detect the ground. You can adjust it by moving the Position of the Transform component.



The proper placement of the Ground Check.

Last is the Trigger. It is a trigger-only collider that that if it touches a game object with a trigger-only collider that is tagged “Ladder” (Tagging and Layer will be discussed in the Customization Chapter), the character would lose the platformer gravity, making it able to climb ladders, or vines, or even swim on water, as long as the tag is set as “Ladder” (see Customization Chapter). Now let’s assign the player character to the Player Character slot. The Ladder Script is attached to the Trigger game object to do the functions stated beforehand.

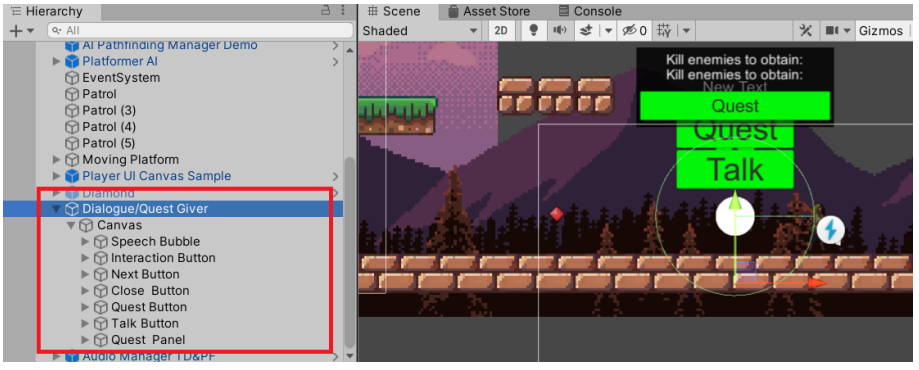
The Ladder Climb Script attached to the Trigger

Patrol Points (for Moving Platforms, extended)

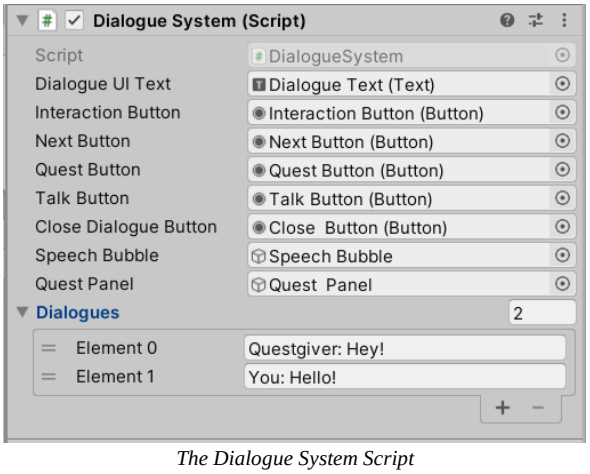
Like the patrol points for the AI, the patrol points for the moving platforms are specified points in the map where the platforms can move from position to position. If you want to change its position, you can do so by. Changing the Position of its

Transform component. Preferably do not overlap the patrol points' positions with each other or with the AI patrol points. It would cause unwanted collisions inside the game and may stick the objects in the weirdest ways imaginable. To also help in distinguishing, name your AI patrol points in the hierarchy as "AI Patrol Point" while the moving platform patrol points as "Moving Platform Patrol Point" and add a number at the end of the name to further distinguish each points to avoid confusion.

Dialogue and Quest Giver (Platformer, can also be used in Top-down)



The anatomy of the Dialogue and Quest Giver



The Dialogue System Script

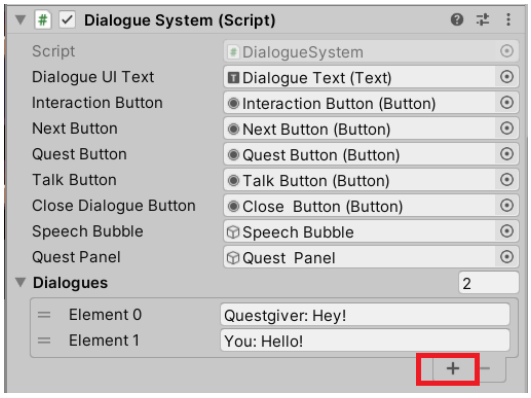
The Dialogue and Quest Giver game object, as the name suggests, handles both the questing and dialogue. As we can see, the game objects that we need to assign are conveniently named the same to avoid confusion in assigning multiple parts. So, let's assign each individual part (not needed if you don't modify the script and template).

To detect if the player is on the vicinity of the Dialogue and Quest Giver game object, it uses the similar detection style of the AI by using trigger-only colliders (see AI under Chapter 1 for reference). Like the AI, you can adjust the size of the collider to suit to your preference.

A quick rundown for the function of the buttons:

- Interaction Button = opens the Interaction Panel which you can choose if you want to talk or know the quest details.
- Next Button = the button for the next dialogue function.
- Quest Button = when pressed, opens the Quest panel which shows if the quest is still ongoing, or if you have the item needed to complete the quest, or if you have already completed the quest (questing will be discussed in the Customization Chapter)
- Talk Button = when pressed, opens the speech bubble and the dialogue which you can read.
- Close Dialogue Button = if the dialogue reaches its end, this button appears and can close the dialogue.
  Other parts will be discussed on the Customization Chapter

Now, for adding new sentences for the dialogue, click the Add icon (in red) to add a new entry or multiple entries. You can then type the entries and it will automatically be added into the game. To make the dialogue less confusing, put names before every sentence that will be shown just like the example below:



Adding new entries for the Dialogue

Questing will be discussed on the Customization Chapter

Audio Manager (extended)

While the previously discussed audio manager only handles the sound effects to play, this audio manager handles the controls for the settings. Clicking the circle with a dot in the center (which will be called as Search from this point onwards) will suggest the Audio Mixer pre-made inside GDSP. It should be the one to use unless you created your own. Clicking the Search for both the sliders would suggest the sliders in the scene. Just assign accordingly, since the names match to the slots specified. Lastly, assign the Exit Button from this location:

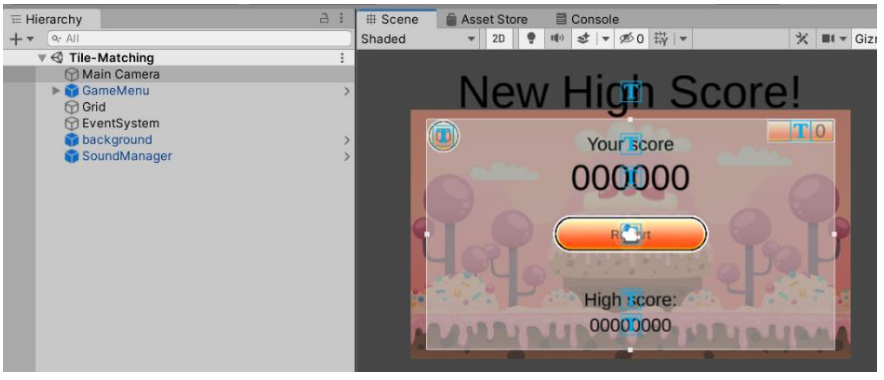
- Player UI Canvas Sample
  - Settings

- Exit

…to complete the setup. It is done so for the purpose of the exit button being selected first for the console users for easy navigation.

The extended audio manager.

Chapter 3
Tile-Matching



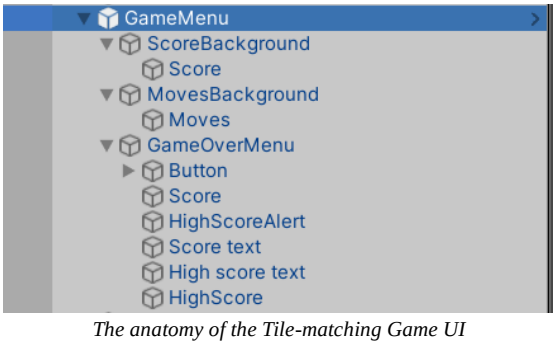
The anatomy of a Tile-matching template.

The tile-matching template borrow its roots from the famous tile-matching games such as Bejeweled, Candy Crush, and the likes. The template contains the most basic function of the game, which is to match tiles of the same color. The template is similar to the Candy Crush gameplay which the user has limited moves and would game over if the moves are zero. The template contains the following game objects:

- Game UI
- Tile Grid
- Background Image
- Sound Manager
- Tile Script
- Camera (already discussed in the previous chapters)

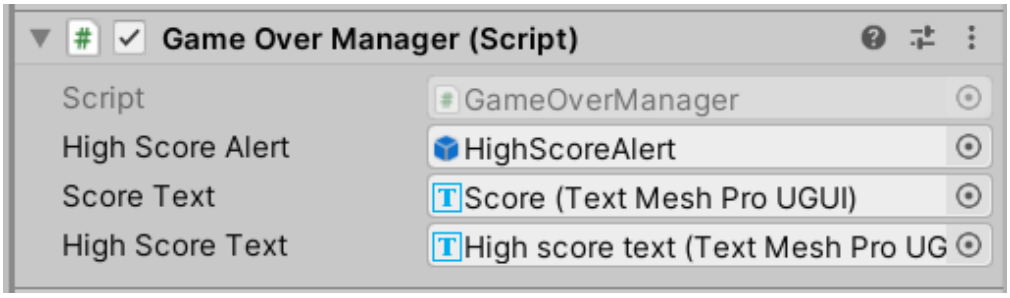
Game UI

First is the Game UI named GameMenu. It contains the UI needed to be displayed when playing the game.



The anatomy of the Tile-matching Game UI

The Score Background is the image that holds the Score Text. The Moves Background holds the Moves Text. Lastly, the Game Over Menu holds the Restart button, The text that says “High Score”, the text that says “Your Score”, and the text that shows “New High Score!”. Then, the text that changes based on your score and the high score (Score and High score text). The Game Over Menu game object holds the functions for the UI named Game Over Manager.



The anatomy of the Tile-matching Game UI

The High Score Alert is the text that says, “New High Score”. The Score Text is text that changes, not the one that says score, same with the High Score Text. Since you already know how to assign game objects, the location of which to assign will be shown below:

For High Score Alert:

- GameMenu
    - GameOverMenu
        - Score

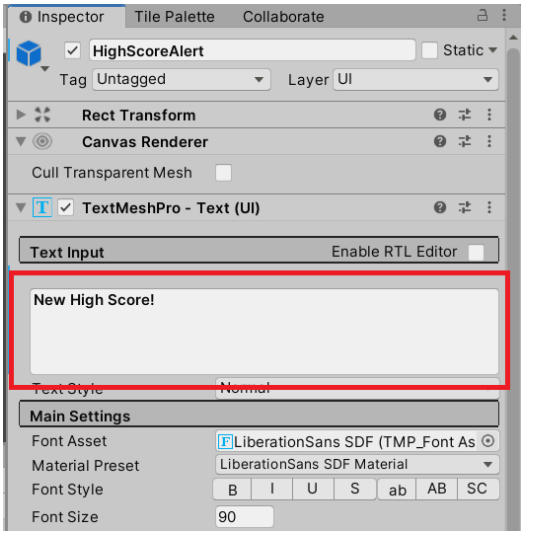
For Score Text:

- GameMenu
    - GameOverMenu
        - HighScoreAlert

For High Score Text:

- GameMenu
    - GameOverMenu
        - High score text

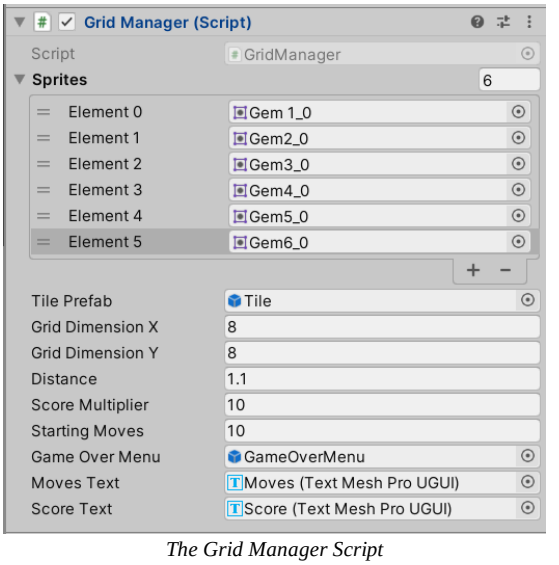
For the other texts inside the GameOverMenu, you can change the texts that you want to show just by clicking the game object and going to the TextMeshPro – Text (UI) component. You can also change the font, style, size, color, spacing, alignment, and more.



The location (in red) for changing the texts.

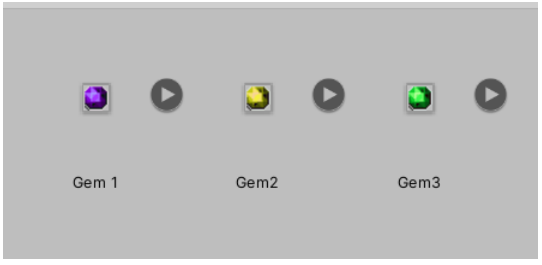
Tile Grid

Now, let's move the Grid. The Grid game object contains the Grid Manager script. The script manages all the functionalities of the game.

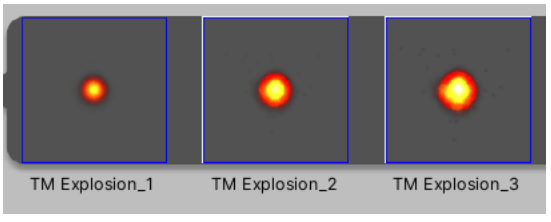


The Grid Manager Script

The first thing that we see is the Sprites array. These are the images that will be used in the game. To add an image to the array, you can use any image that you like, but to be more precise, use images that fill the space edge to edge. You can then get the images from the Assets window.

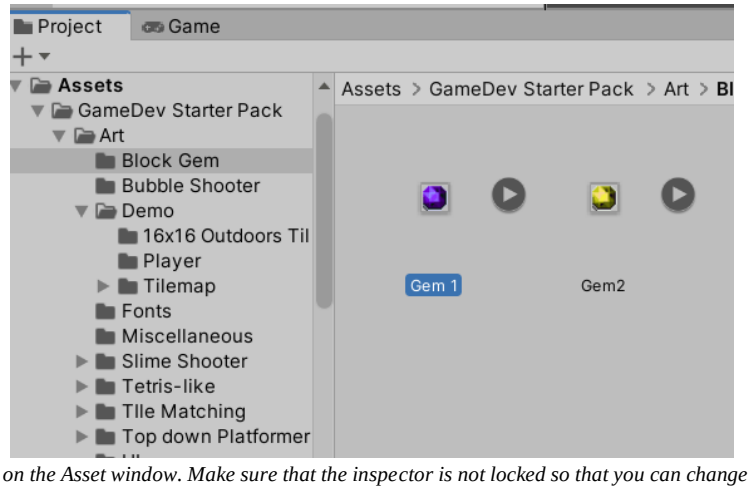


An image with filled space edge to edge. We can see that the image are packed and enclosed and there are no spaces across the images except for the edges.

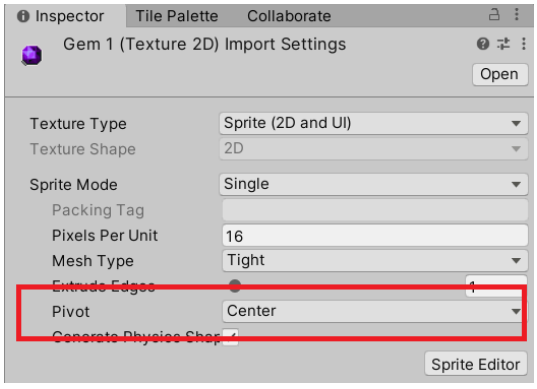


The opposite of the above, which shows a lot of space on the sides.

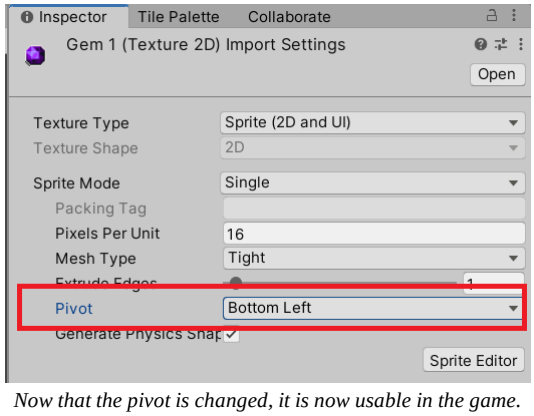
By doing so, the tiles will be properly placed in the grid. To add to this, you also need to set the pivot of the image to be on the bottom left. If you want to use the same asset for different purposes, you can just duplicate the image so that you can change its pivot. The reason we do this is so that the player can have a larger room to click compared to the pivot being in the center. To change the pivot of the image, follow this:



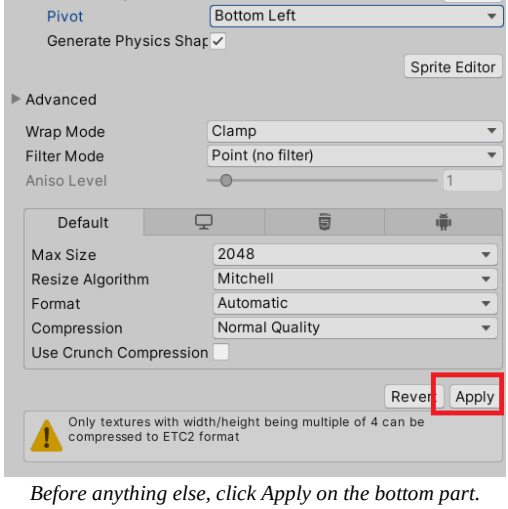
Select an asset on the Asset window. Make sure that the inspector is not locked so that you can change the settings of the image. Upon clicking, the Inspector will show the settings for the image.



We can see that the Pivot is center, which is the default in Unity. We need to change that to Bottom Left for it to be usable in the game.



Now that the pivot is changed, it is now usable in the game.



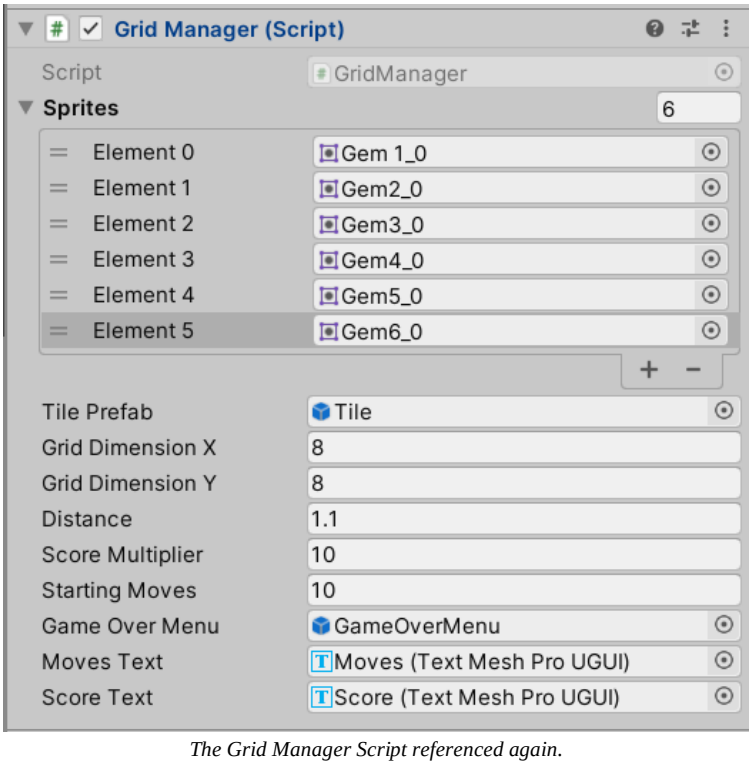
Before anything else, click Apply on the bottom part.

We already know how to add entries to the array, so we can now just add the edited image to the Sprites array. These would be the images that will show in the grid if the game is started.

We can now move to the variables that we can freely change to customize the game. First is the Grid Dimension X and Y. This sets how tall or wide the grid would be. X is for the across, Y is for down if we use crossword terminologies. The Distance would be how far apart are the tiles to each other. You can freely adjust this but 1.1 is the sweet spot for this configuration. Next is the Score Multiplier. It is the number that multiplies to how many tiles were you able to remove. If you remove 3, 3 is then multiplied to the number specified here. For example, the multiplier is 15, and you cleared 4 tiles, your score would be 60.

Last for the easy to modify variables is the Starting Moves. This is how many matches the player can make before hitting game over. The recommended is the bigger the size of the grid, the higher the move count, and vice versa. But you can then flip it, giving many moves to a small grid and few moves to a big grid. Here comes the issue. The player might run out of moves if the grid is too small, but he/she still has many moves. The game would not be balanced.

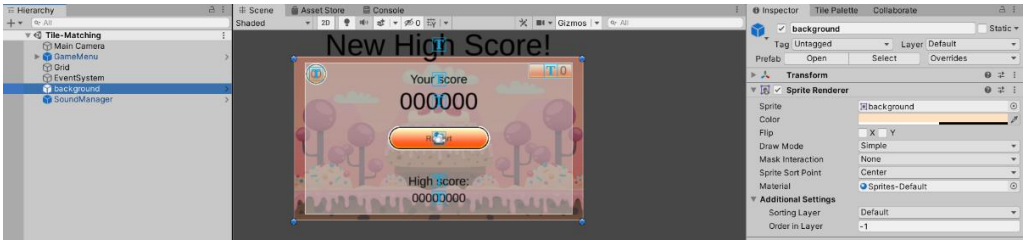
The Moves and Score Text are located at the Game UI, and can easily be assigned here. The Tile Prefab will be discussed later.



The Grid Manager Script referenced again.

Background Image

The next part is the background image. It is simply just a background image. Nothing more, nothing less.

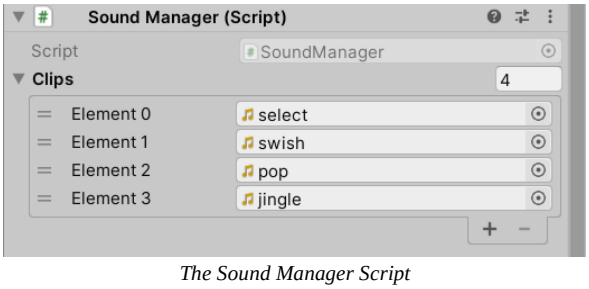


The Background Image

Sound Manager

The sound manager handles the sounds that play in the game. While you can edit this one, if you are not yet experienced with Unity, we suggest that you do not touch the Sound Manager. Despite that, we can explain how this works.

There is a Clips array that contain the sound effects needed for the game. The clips are named select, swish, pop, and jingle. The audio clips will be played depending on what is happening in the game. If you want to change the clips, rename your audio clips from the Asset window to the desired effect you want to swap. For example, if you want to change “select” and the audio clip you want to replace is named anything other than select, rename it to select, delete the old select entry and add your new one. The order will not matter.

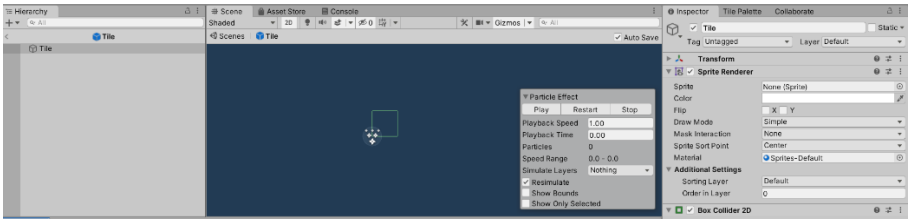


The Sound Manager Script

Tile Script

The Tile script holds all the functionalities for the pressing, swapping, destroying, and filling the empty spaces after clearing. We can find the Tile Prefab in this location:

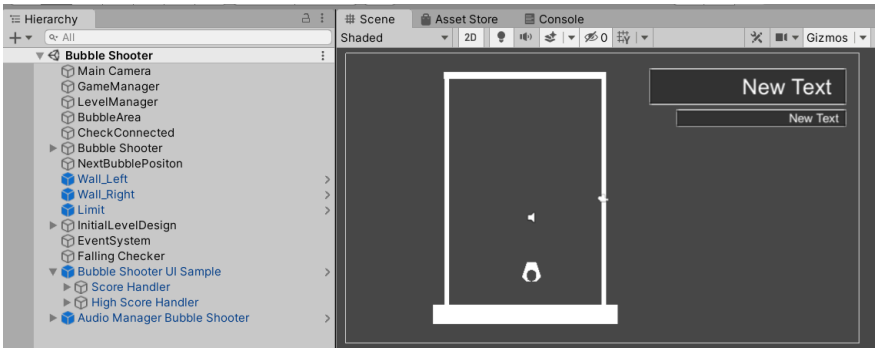
- GameDev Starter Pack
  - Functions
    - Tile-Matching
      - Tile



The Tile Prefab

This is the prefab that will be assigned to the Tile Prefab slot. As we can see inside Unity, we cannot find the Tile script attached to the prefab. The Grid Manager is the one that attaches the Tile Script to each individual spawned tiles, which simplifies a lot of things since the user doesn't need to assign components to the Tile script anyways.

Chapter 4
Bubble Shooter



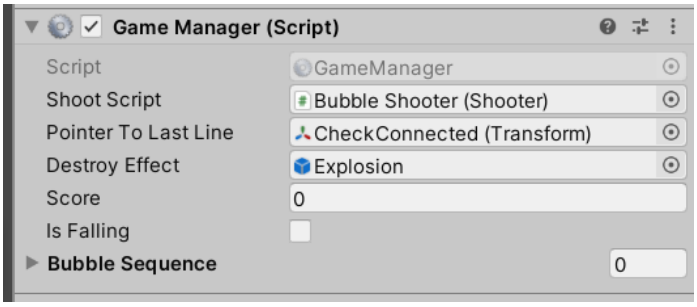
The Anatomy of the Bubble Shooter Template

The Bubble Shooter template borrows its roots from Puzzle Bobble, a classic bubble shooter game. The template allows the user to play a level of bubble shooter, in which the player shoots similar-colored bubbles to pop them. The connected similar colors should be minimum of three to register as a score. Bubbles will also fall if there are not connected to other bubbles. For example, you hit a cluster of red bubbles with a red bubble and another-colored bubble is attached to that cluster, the other bubble will fall of and will still count as a score. The bubble shooter template contains the following:

- Main Camera (already discussed on previous chapters)
- EventSystem (already discussed on previous chapters)
- Bubble Shooter Game Manager
- Bubble Shooter Level Manager
- Bubbles
- Bubble Area
- Check Connected
- Bubble Shooter
- Next Bubble Position
- Walls
- Falling Checker
- UI
- Audio Manager

Bubble Shooter Game Manager

The game manager handles the processes happening during the game. The game manager's purpose is to keep track of the score, check the position of the top line, knowing if the shooter can shoot, the effect during the popping of bubbles, check if there are falling bubbles, and knowing the sequence of bubble that was shot. It is attached to the Game Manager game object.

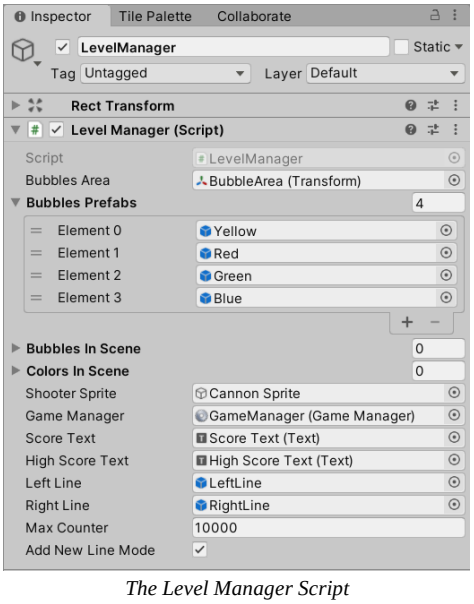


The Game Manager Script

The Shooter Script slot is for the Bubble Shooter (to be discussed later). Assign the Shooter here. The Pointer To Last Line checks the position of the top line (to be discussed later). This determines the location of where a new line will be spawned. As this template uses the hexagon grid, it also determines if the line to be added is even or odd. The Destroy Effect is a prefab that is spawned when the bubbles are popped when matched (see Customization). The prefabs can be found on GameDev Starter Pack > Templates and the Explosion prefab can be found. It can also be easily searched (reference to Search discussed previously). The Score handles the score the player has collected during play. The fixed score is 10 points. The Is Falling checkbox should not be checked since it is just a reference. The Bubble Sequence should also be not touched as it only lists the sequence of bubble that was shot.

Bubble Shooter Level Manager

The Level Manager handles the setup and some of the on-play functions of the template.



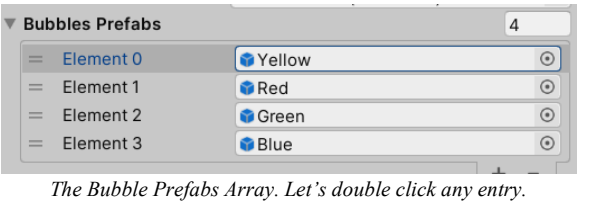
The Level Manager Script

The Bubbles Area is an empty game object that when on-play, is populated by the bubbles (which will be discussed later). The Bubbles Prefabs array is an array that

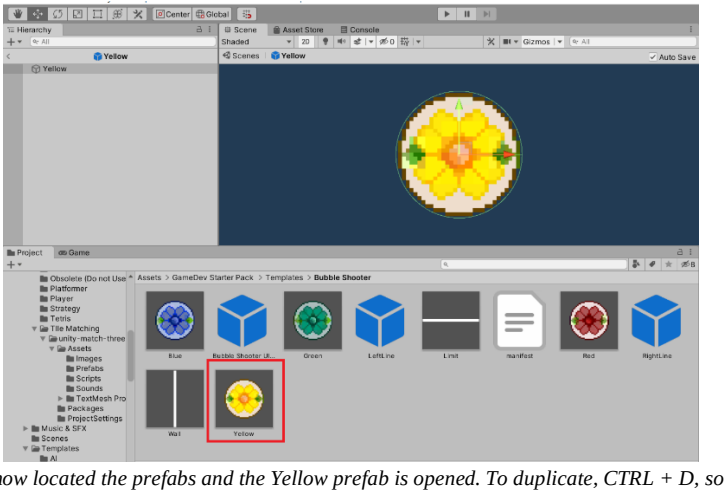
handles the bubbles that will be spawned in-game (which will be discussed later). The Bubbles In Scene and Colors In Scene arrays are arrays that holds, as their name implies, the bubbles and colors in scene. You should not put any entry in these arrays. The Shooter Sprite is basically the child of the Bubble Shooter game object which has the Sprite Renderer component (to be discussed later). The Game Manager is the Game Manager game object. The Score and High Score texts are parts of the UI that show, as their name implies, the current score and the high score (to be discussed later). The Left and Right line prefabs should not be touched. The Max Counter is the number that determines how fast the game adds a new line to the game. The higher the number, the slower the new line adds. The Add New Line Mode checkbox, as the name implies, determines if the game has the Add New Line Mode.

Bubbles

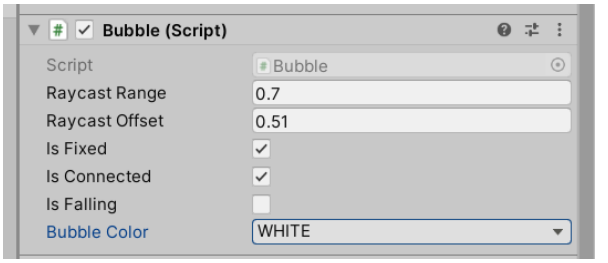
The bubbles are what the player will shoot and the ones to be spawned in the grid. We edit a current bubble or add a new one. To know the location of the prefabs, just double click one of the prefabs to be directed to the location in the Assets window, and at the same time, opens the prefab. Let's try creating a new bubble to be added to the game.



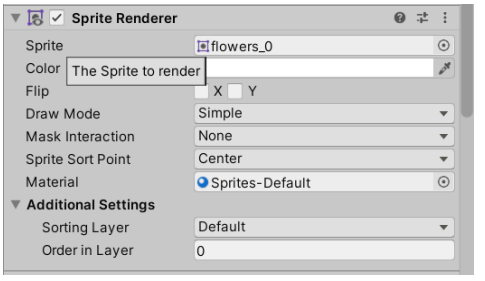
The Bubble Prefabs Array. Let's double click any entry.



We now have now located the prefabs and the Yellow prefab is opened. To duplicate, CTRL + D, so that we can create a new variant. Let's name it White.

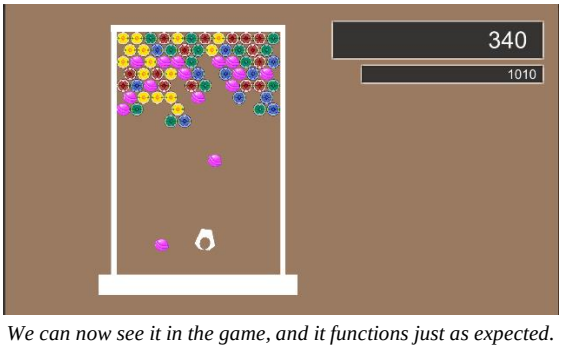


Since we named it White, let's choose the Bubble Color to also be white.



Let's go to the Sprite Renderer component to change the sprite. As we discussed earlier in the package, we can easily change the sprite by clicking the "Search" on the Sprite. Clicking it will show all images available in GDSP. We can choose any image so that it could stand out among the other bubbles. Just make sure that the pivot of the image is set to Center.

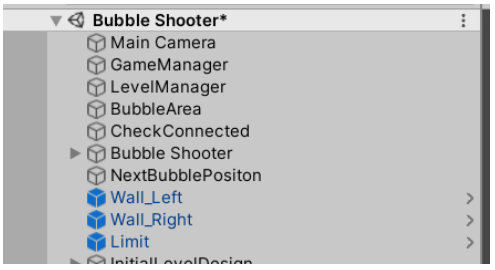
We now have created a new variant and we can add it easily to the game. We already know how to add entries to an array, so we can do that to add White to the game in the slot of the Bubbles Prefabs. The new bubble is now added to the game.



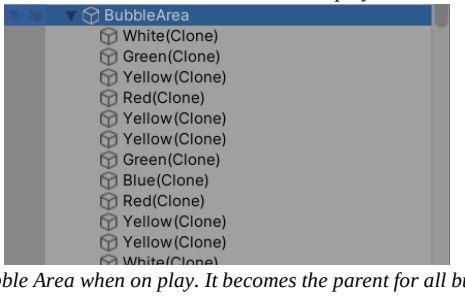
We can now see it in the game, and it functions just as expected.

Bubble Area

The Bubble Area handles all the spawned bubbles. It becomes the parent of all bubbles, and the contents of the Bubble Area can now be manipulated by the Level Manager script to do the miracles of the bubble shooter gameplay.



The Bubble Area when not on play.



The Bubble Area when on play. It becomes the parent for all bubbles.

Check Connected

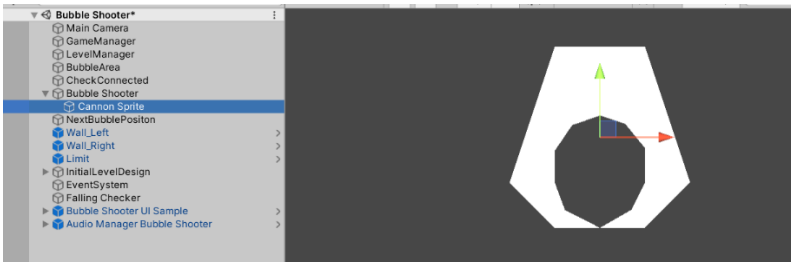
The Check Connected is basically the pointer to where is the last line. There is no other thing that can be told from this. Just don't move it around to not cause errors.

Bubble Shooter

The Bubble Shooter Script

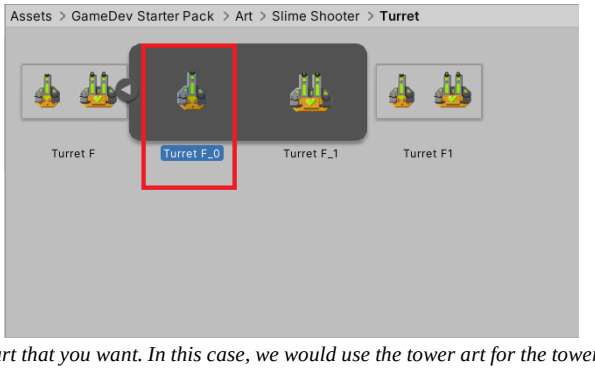
The Bubble Shooter is the cannon that shoots bubbles to the grid. The Gun Sprite is the child of the Bubble Shooter game object (to be discussed later). The Can Shoot checkbox tells if the player can shoot or not. If the cannon is pointed down, it will not shoot. Just don't touch this bit. The Speed is how fast the bubble is shot. You can adjust it to your preference. Just make sure that it is greater than zero. The Is Using Mouse is the checkbox to determine the type of device the player uses. If it is checked, it uses the mouse controls. If not, it uses the console-like controls. The Next Bubble Position is the position to determine where the next bubble is spawned. The current

bubble is the spawned bubble that is seen on the cannon. The next bubble is the one that is spawned on the Next Bubble Position. Just don't assign anything in these slots since they are used for debugging, not for assigning. The Level Manager slot is just the Level Manager in the hierarchy. The Turn Speed is the speed on how fast the cannon turns if using the non-mouse control. You can also set this up on your preference.

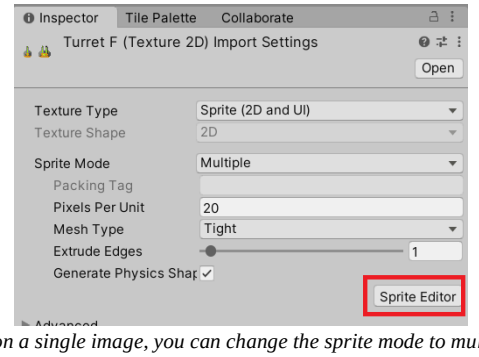


The Cannon Sprite

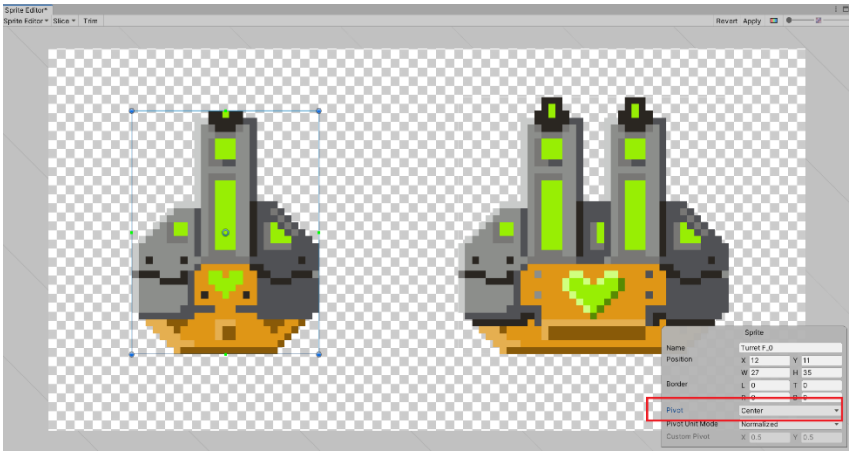
Let's go to the Cannon sprite, which is the child of the Bubble Shooter. This is where the image for the cannon is placed. You can assign your custom cannon art here. But first, let's change the pivot of the cannon art image so that it would spin at the right spot.



Choose the cannon art that you want. In this case, we would use the tower art for the tower defense located in GameDev Starter Pack > Art > Slime Shooter > Turret. We would use Turret F_0. But first, click the Turret F to enter Sprite Editor.

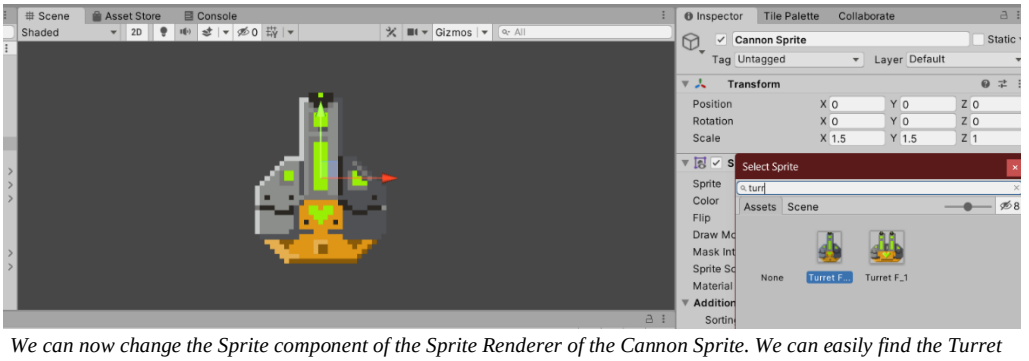


If you have multiple sprites on a single image, you can change the sprite mode to multiple. If not, then just use the default. Now, let's click Sprite Editor.

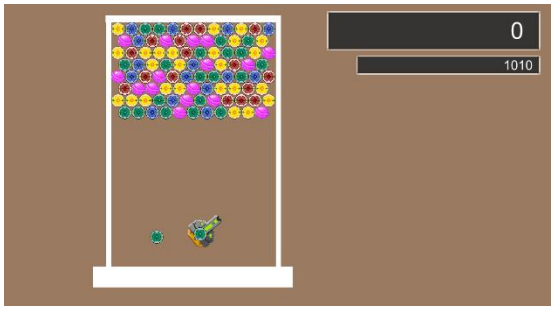


Now, we have entered the Sprite Editor. We can see that the pivot is in the dead center of the sprite. We don't want that. We want the pivot to be placed to the center of the circular part of the tower, which in this case, is the green heart. Set the pivot to Custom so that we can move the pivot point.

Let's only adjust the Y in the custom pivot since we only want it to move vertically. Now that we have placed the pivot in the proper position, we can now change the default sprite in the Cannon Sprite.



We can now change the Sprite component of the Sprite Renderer of the Cannon Sprite. We can easily find the Turret F_0 that we edited by using Search (discussed earlier). We can now test it in the game.



We can now see that the cannon is rotating as intended. Note that GDSP only accepts tower images that are facing on the local north direction, so if you want to use a custom art, make sure that it is facing to the local north direction.

Next Bubble Position

The Next Bubble Position is an empty game object which only holds the position of where you want the next bubble to appear. You can adjust this to your heart's content via the Position in the Transform component attached to it.

Walls

The walls are images that binds the grid altogether. The walls have a Physics2D component that reflects what was shot into it. In the game, it will make the bubble go to the opposite direction. If the walls are not placed, the bubble will travel indefinitely and will crash the game. Just don't change it if you have limited Unity knowledge, but you can change the image that the walls use in the way like changing the sprite of the Cannon Sprite.

Falling Checker

The Falling Checker checks if anything in the scene is falling. This is linked to the Game Manager.

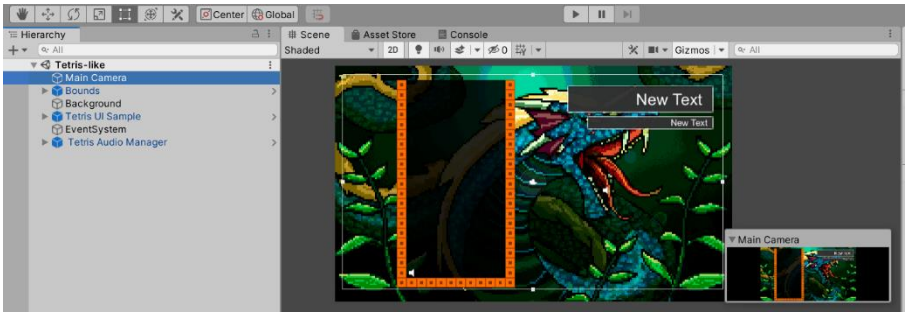
UI

Since you already know how to assign much more complex UI templates, assigning the elements of this is easy. In the Level Manager you can assign the needed UI there.

Audio Manager

The Audio Manager of the Bubble Shooter functions the same as the Audio Manager of the Top-down RPG. So, you can use it for reference.

Chapter 5
Tetris



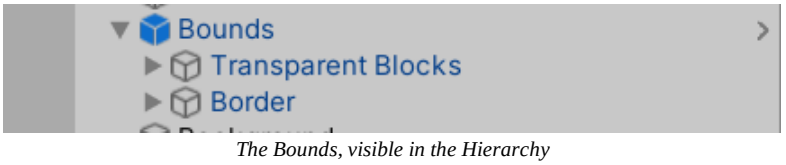
The Anatomy of the Tetris Template

The Tetris template is an imitation to one of the most famous puzzle games of all time, Tetris. The template uses the standard Tetris values, with a grid size of 10 blocks for width and 20 blocks for height. The scoring system is also derived from the standard Tetris values, such as 40 points for 1 line cleared, and so on. Similar to the real Tetris game, the template also has 7 pieces, namely: J, L, Long, S, Square, T, and Z. The Tetris template contains the following:

- Main Camera (already discussed on previous chapters)
- EventSystem (already discussed on previous chapters)
- Game Bounds
- Background Image
- Game UI
- Audio Manager
- Tetris Pieces

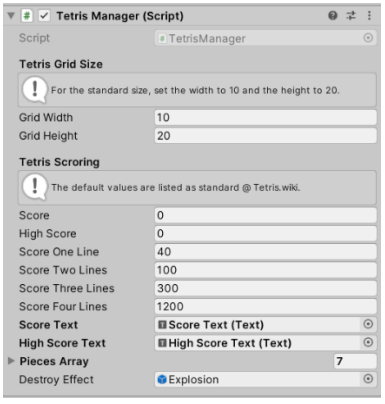
Game Bounds

The Game Bounds is basically the board of the game. The game object houses 2 game objects. The first one is the Transparent Blocks which contains 200 individual blocks that serves as the background of the board. The second one is the Border in which, as the name implies, houses 53 individual blocks to create the border. These individual images can be customized in both art and color. Make sure that what you change to are square images.



The Bounds, visible in the Hierarchy

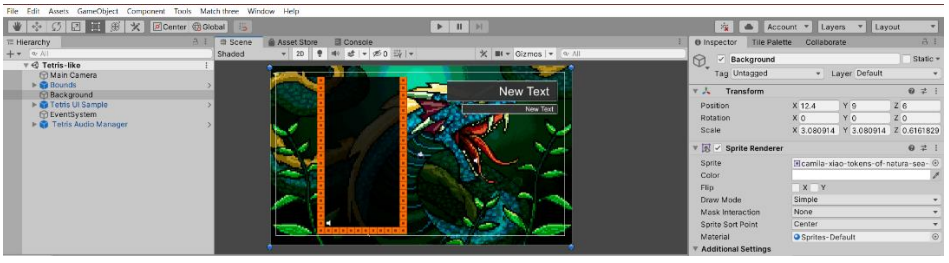
The Game Bounds also contains the Tetris Manager script that handles all processes inside the game. The script handles everything from scoring, spawning effects and to setting the UI. The script is already intuitive, so you can easily assign different values into it.



The Tetris Manager Script

Background Image

The next part is the background image. It is simply just a background image. Nothing more, nothing less. Before we forgot, you can also add an image background to the bubble shooter template (in which GDSP will have placed).



The Background image of the Tetris template

UI

Since you already know how to assign much more complex UI templates, assigning the elements of this is easy. In the Tetris Manager, you can assign the needed UI there.

Audio Manager

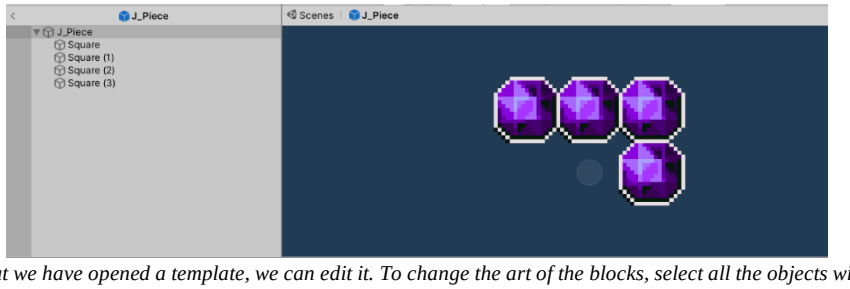
The Audio Manager of the Tetris functions the same as the Audio Manager of the Top-down RPG. So, you can use it for reference.

Tetris Pieces

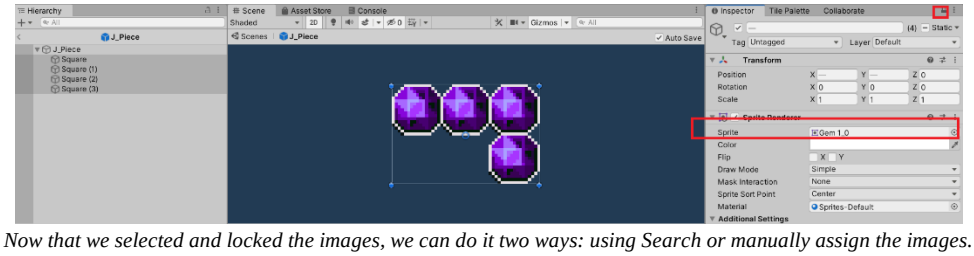
The Tetris pieces are the ones that fall and the ones that you need to build into structures that you can score into. You can edit these pieces to your liking. First, go to the Bounds game object and locate the Tetris Manager script. At the bottom of the script, you can find the Pieces array. Like editing the tiles in tile-matching, you can double click any entry on the array to find its location. For a manual approach, you can find the pieces in:

- GameDev Starter Pack
    - Templates
        - Tetris

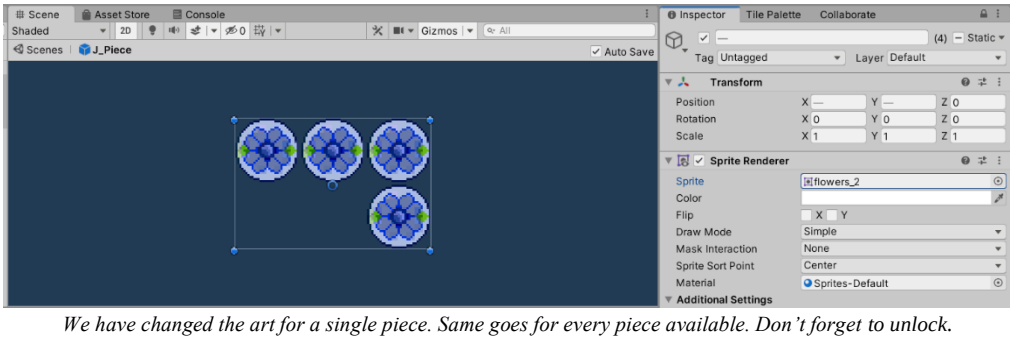
You can also find the template for the Bounds if you want to edit it. Now, let's try editing a piece.



Now that we have opened a template, we can edit it. To change the art of the blocks, select all the objects with the name "Square" as each one holds the images that can form into the piece that you can see. Make sure to lock after selecting the 4 game objects.

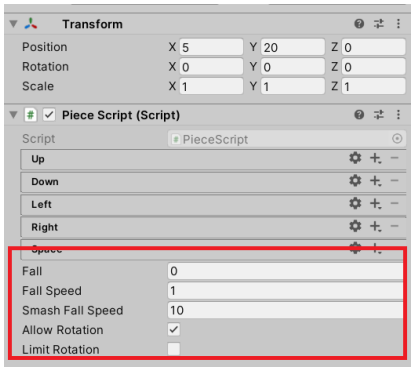


Now that we selected and locked the images, we can do it two ways: using Search or manually assign the images. Let's just use the Search in this bit.



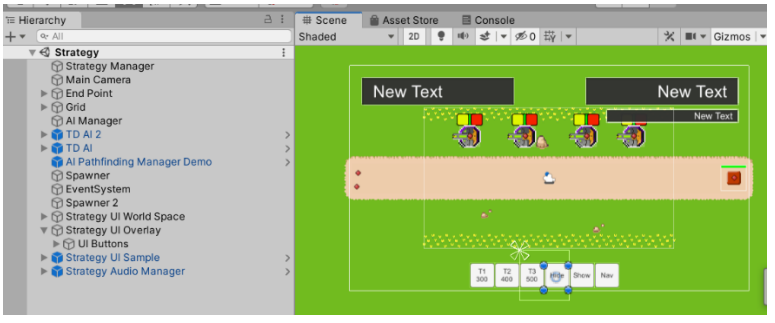
We have changed the art for a single piece. Same goes for every piece available. Don't forget to unlock.

Now that we have customized the art, we can also customize how fast the piece falls. We can adjust the Fall Speed. This is the default fall speed. The smash fall speed is the speed if we pressed the smash button or key bind. All pieces have different settings on the Allow Rotation and Limit Rotation. Just don't touch those when customizing.



We have changed the art for a single piece. Same goes for every piece available. Don't forget to unlock.

Chapter 6
Strategy (Tower Defense)



The Anatomy of the Strategy Template

The Strategy template is modeled around the tower defense game. Enemies would go through a path in which the towers that the player builds would shoot at the enemy. If the enemy reaches the end point, the player will lose a life. If enough enemies pass through the end point, it's game over. The following are the contents of the Strategy template:

- Main Camera (already discussed on previous chapters)
- EventSystem (already discussed on previous chapters)
- Strategy Manager
- Grid (already discussed on previous chapters)
- End Point
- Placebo AI Manager
- Real AI Manager (already discussed on previous chapters)
- World Space UI
- Overlay UI
- Audio Manager

Strategy Manager

The Strategy Manager holds the Camera Controller script which controls the placing of the towers and the navigation inside the game. The game controls are for console type, so placing on-screen buttons for touch is a must (see Customization chapter). The script only contains the array of the tower placements found in the Strategy UI World Space (which will be discussed later).

End Point

The End Point is a game object with an attached box collider to register damage. It also contains the Health Counter Script which holds the information of the current health of the end point which determines the game over condition.

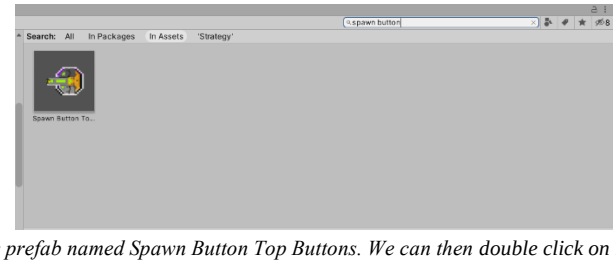
Placebo AI Manager

In the real AI Manager, it handles the pathfinding capabilities of the AI present in the scene, but in the context of the Top-down RPG, the real AI manager gets turned off if the player is not detected by any enemy. To combat turning off the real AI manager since the manager is always used by the enemies, we have added a placebo AI manager which acts as the turning on and off of the AI. Since it doesn't contain anything, it doesn't affect the game. The issue comes if there are no AI Manager assigned in the AI Master Script, since the AI master script needs an AI manager to hold into. The real AI manager is unaffected, while the enemy AI does what it does without any issues.

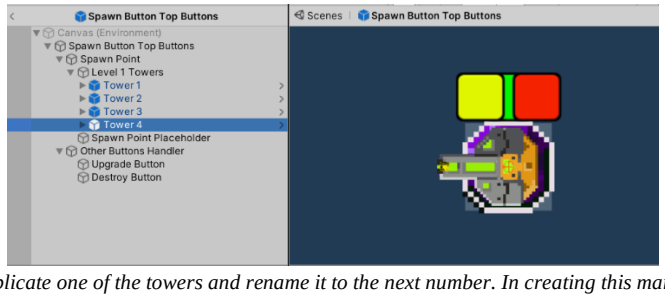
World Space UI

The Anatomy of the World Space UI

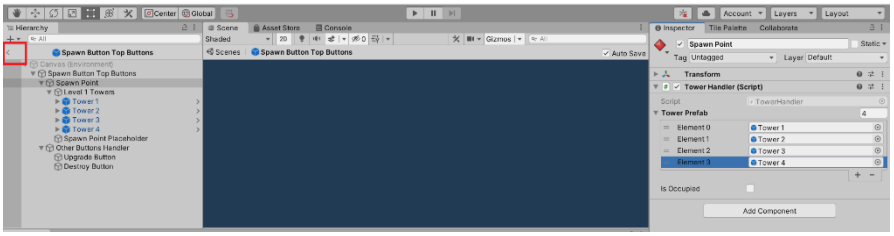
The World Space UI are basically the locations of the towers. These UI are placed directly on the map instead of being an overlay (see Overlay UI). These individual spawn points can be placed anywhere in the map. We can see that the children of World Space UI are prefabs. If we open the prefab, we can edit every Spawn buttons. Let's try adding a new tower for the game. This will be linked to the next part (Overlay UI).



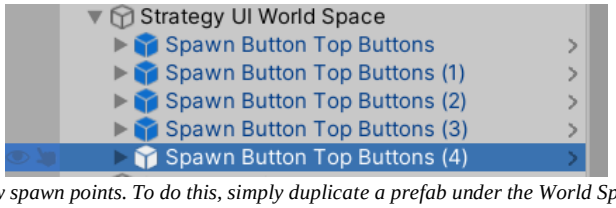
Let's search for the prefab named Spawn Button Top Buttons. We can then double click on it to open the prefab.



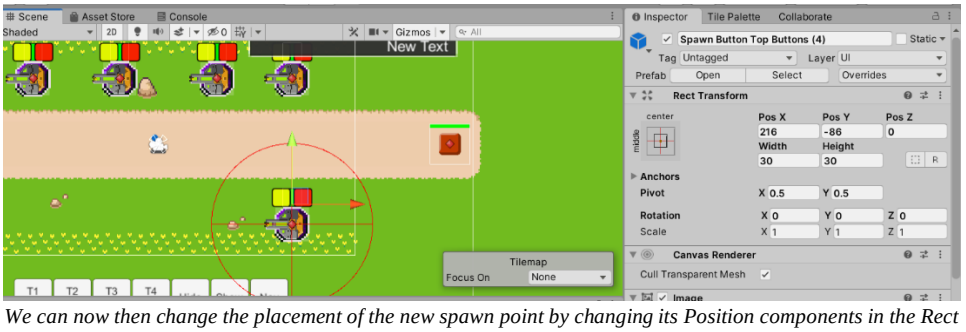
We can then duplicate one of the towers and rename it to the next number. In creating this manual, the number towers are 3. By adding another, it is now 4. In your case, you may start at 5. The naming matters very much in this context, so please follow the naming convention.



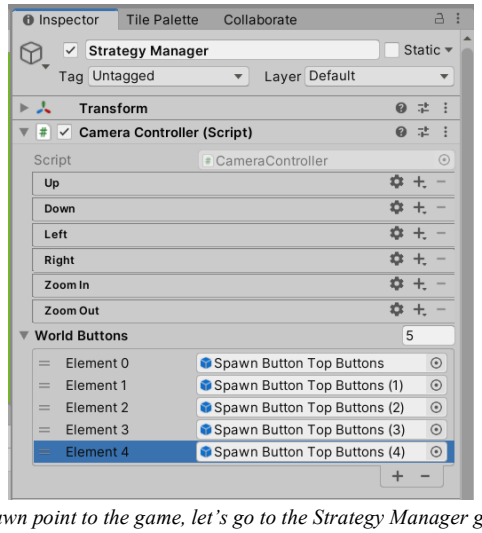
Now, let's click the Spawn Point game object. We can then see the Tower Handler Script. We can now then add the Tower 4 by dragging and dropping it to the Tower Prefab array. This Prefab is now ready as we fully integrated a new tower to the game. We can now then close the prefab by clicking the arrow going to the left (in red). In modifying the towers, see Customization Chapter.



We can also add new spawn points. To do this, simply duplicate a prefab under the World Space UI game object.

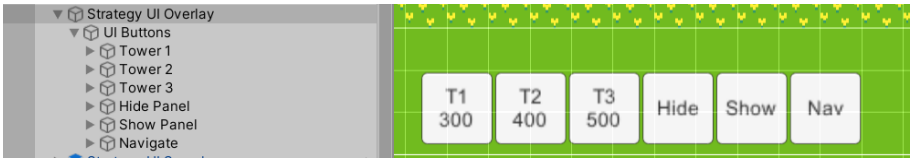


We can now then change the placement of the new spawn point by changing its Position components in the Rect Transform.



To properly add this new spawn point to the game, let's go to the Strategy Manager game object and add the new spawn point in the World Buttons array. Now, the new spawn point is integrated to the game.

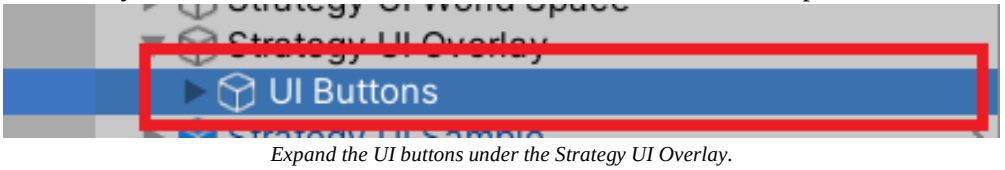
Overlay UI



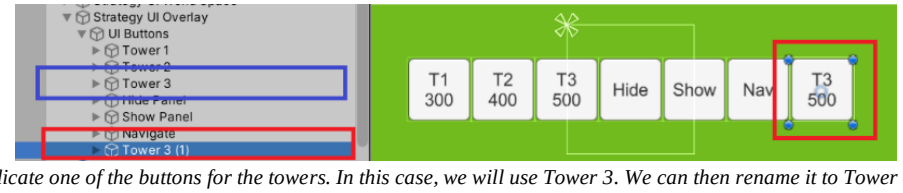
The Overlay UI

The Overlay UI contains the buttons used to buy the towers. The names for the buttons must be similar to the names of your towers to not get confused when playing the game. The template has 3 towers to play with, so the game only shows 3 towers that you can buy (T1, T2, and T3). The Hide Button hides the buttons for the towers that you can buy to simplify the game UI. The Show button does the opposite of the Hide Button. The Nav (Navigate) is the button used for when the player is upgrading.

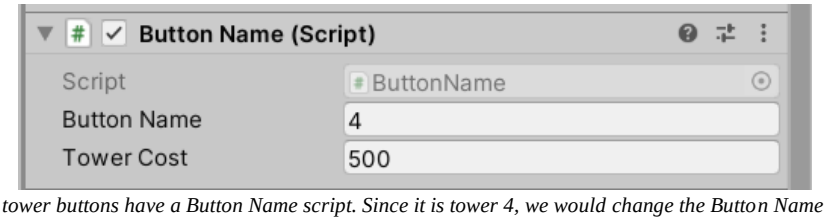
If you want to add a new button for a tower, follow these steps:



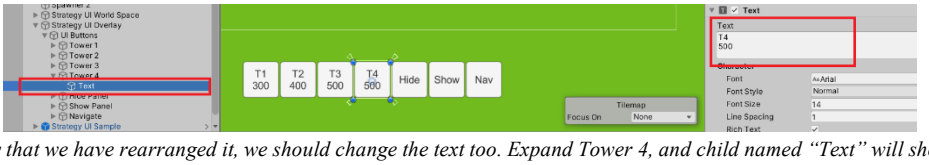
Expand the UI buttons under the Strategy UI Overlay.



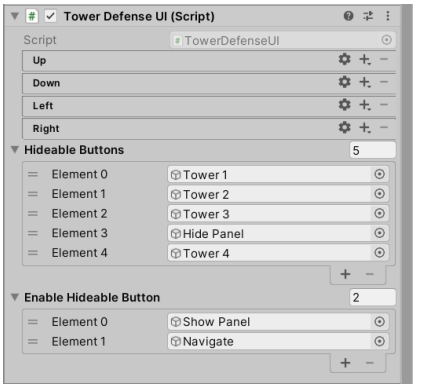
Duplicate one of the buttons for the towers. In this case, we will use Tower 3. We can then rename it to Tower 4. We can move its position below Tower 3 in the hierarchy to properly display the buttons.



The tower buttons have a Button Name script. Since it is tower 4, we would change the Button Name to 4.



Now that we have rearranged it, we should change the text too. Expand Tower 4, and child named "Text" will show. The Text game object has a Text component. We can edit that. We can change it to T4 so that it would also show T4 in the UI.



After that, we can now then go to the UI Buttons Child of the Strategy UI Overlay. We can then add Tower 4 in the Hideable Buttons array. It is now linked with the Tower 4 that we made earlier. And that's it for the Overlay UI buttons for this template.

UI (part of Overlay, but only the Scores and Currency)

Since you already know how to assign much more complex UI templates, assigning the elements of this is easy. In the Strategy UI Sample, you can assign the needed UI there.

Audio Manager

The Audio Manager of the Strategy functions the same as the Audio Manager of the Top-down RPG. So, you can use it for reference.

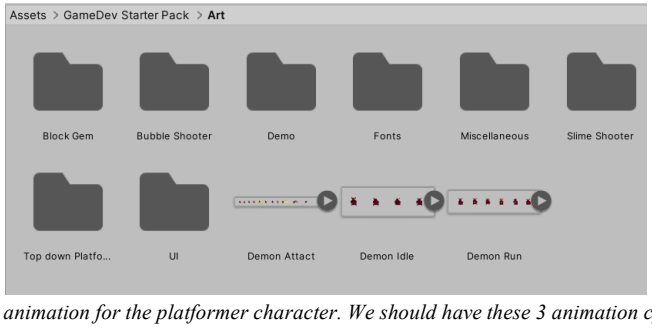
Chapter 7
Customization

This chapter holds everything that we passed up since each part in this chapter needs its very own highlight. Here are the ones to be highlighted:

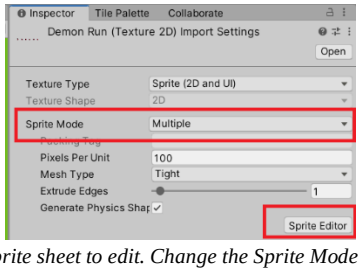
- Automatic Animations (for Player Character, AI agents, and effects)
- Towers
- Projectiles
- Dialogue & Questing
- Items
- Input Binding
- On-screen Buttons
- Creating Tilemaps and Palettes
- Adding Obstacles for AI
- Parallax
- Tags and Layers

Automatic Animations (for Player Character, AI agents, and effects)

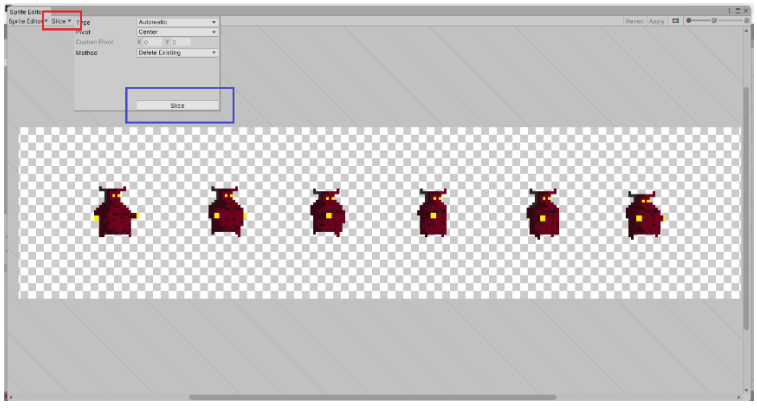
The automatic animation function is the product of our laziness to animate things in frame by frame. By doing this, you can easily animate frame by frame animations without even using Unity's animation window. This function is also used in effects and the AI (which can be found in the Animation Handler Script). To use this function, follow these steps:



Let's try editing the animation for the platformer character. We should have these 3 animation cycles, namely: Idle, Move/Run, and Attack. In this case, we will use this Demon animation set. If you have a sprite sheet like this, follow the next steps. If you have individual sprites, you can move ahead.

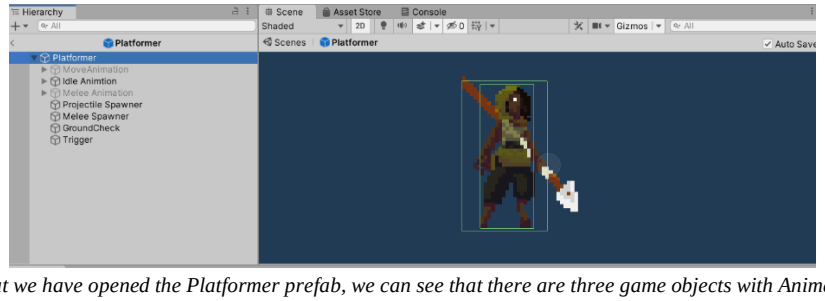


We have selected the Demon Run sprite sheet to edit. Change the Sprite Mode from Single to Multiple and head to Sprite Editor.

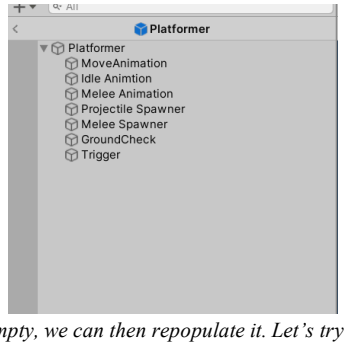


The Sprite Editor is now open. We can now then slice the sprite sheet. Click the Slice (in red). If you know the dimensions of your sprite sheet, you can change the type. But if you want it to be simplified, just leave it as is in Automatic. Let's then click Slice (in blue).

We can now click it as individual sprites. In the Asset window, we can also see that the sprite is cut into multiple parts. We can then do this to the other sprite sheets. Let's now open our Platformer Character temple so that we can change it with this.

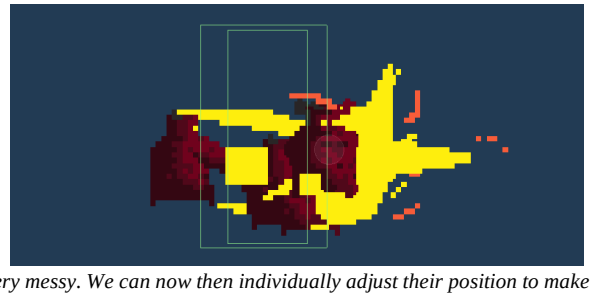


Now that we have opened the Platformer prefab, we can see that there are three game objects with Animation in name, which the name corresponds to the cycles that we prepared. Let's Expand each game objects and delete all its contents.



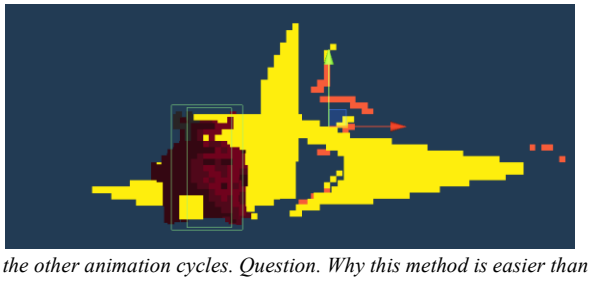
Now that every Animation cycle is empty, we can then repopulate it. Let's try adding the Melee animation. Locate your Attack animation cycle and the sprites that you need and drag and drop it to the Melee Animation game object.

Add the sprites one by one to not generate an animation that you do not want. Now that we repopulated it, let's select every sprite and set all position to zero and scale to 1. It will look like the picture above.



We can see that it is very messy. We can now then individually adjust their position to make the animation smooth. But first, click the topmost game object to see the framing. There is a guide there to where you would place your character. As we can see in the framing, our character wasn't even properly placed or scaled. The position of the main body of every individual sprite should at least be placed on the innermost rectangle.

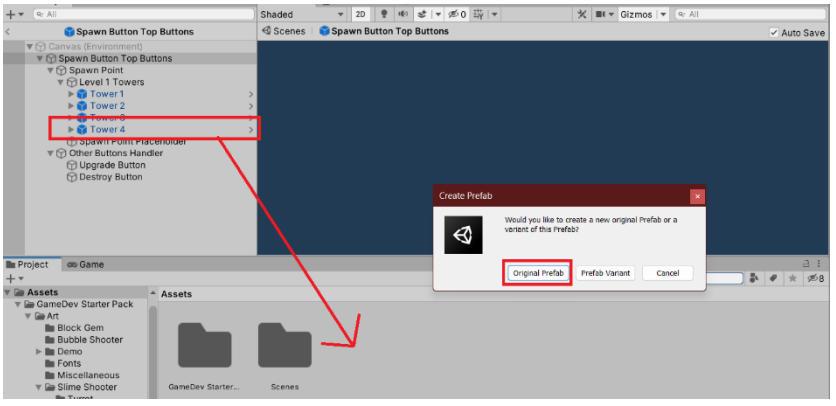
We can then readjust each scale and position to properly place each sprite. To make it simpler, turn off other sprites that you don't need to adjust yet or finished adjusting so that you can easily see the framing. For easier placement, you can use the tools, especially the Move and Scale tool seen on the picture above. After readjusting, it should look something like the image below.



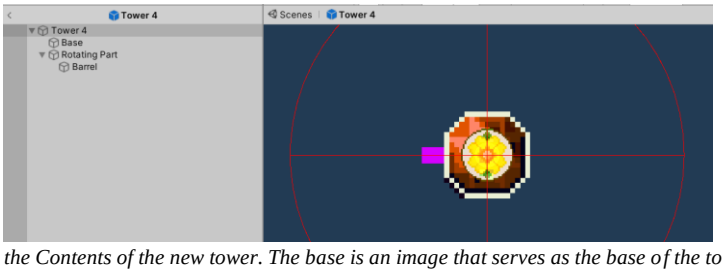
We can do this again to the other animation cycles. Question. Why this method is easier than the Unity's animator? In Unity's animator, you still have to do this tedious placement process, with the added complexity of setting keyframes, which just takes a lot more time. Not to mention, you still have to properly place the animation tree. In this method, you don't have to do that anymore. After doing that, the character animation is now changed and is viewable in game.

Towers

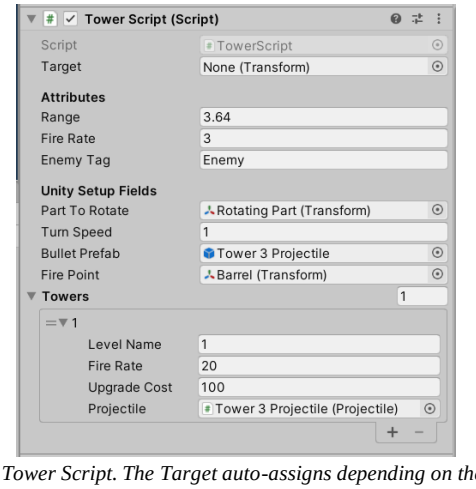
The Tower function is not only used by the tower, but also, the attacking AI in the game. Let's try editing the one that we added earlier, the Tower 4. Let's open the Spawn Button Top Buttons prefab to start the process.



Expand the Level 1 towers and Select Tower 4. Drag and Drop it to the Asset window to create an new prefab. Select Original Prefab. Then, open the new prefab.

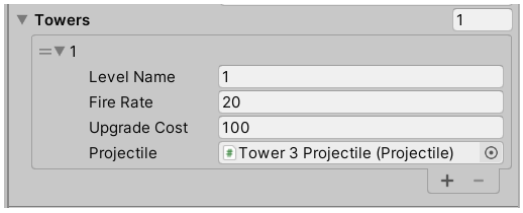


We can now edit the Contents of the new tower. The base is an image that serves as the base of the tower (the orange gem) The base is not needed in the AI shooter since the AI animation already serves as the base of the shooter. The rotating part is basically the rotating part of the tower (the yellow bubble). The barrel is the pointer of where the projectile is shot. To change the tower, click the Rotating Part.



The Rotating Part handles the Tower Script. The Target auto-assigns depending on the enemy tag assigned. If it is a tower defense game, the entry on enemy tag should be Enemy. If the AI is using the tower, the entry on the enemy tag should be Player. The Range would be how large the range of the tower is. The Fire Rate is how fast the tower shoots projectiles. The higher the number, the faster the shooting. Turn Speed is how fast the tower turns when shooting. The

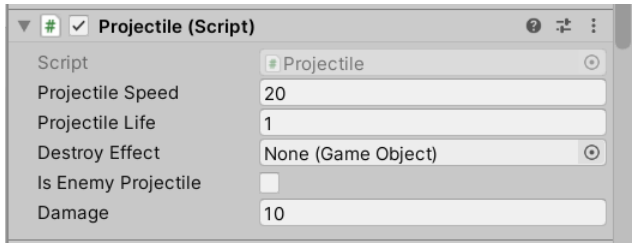
higher the number, the faster the turn speed. Assigning the projectile does not matter yet at this point, as we can assign the projectile in the Towers array.



You can assign tower upgrades here. When adding a new level, click the plus (+) icon then name the Level name the next as the preceding one. The fire rate is like the fire rate. The Upgrade cost is how much is the cost for the upgrade. The Projectile will be discussed next.

Projectiles

The projectile function is used in every projectile in the game. Even the melee attack of the player is a projectile. To edit a projectile, let's locate them first in GameDev Starter Pack > Templates.

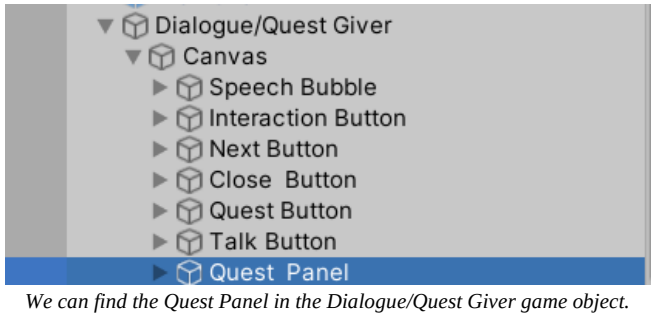


The Projectile Script

The projectile speed determines how fast the projectile travels. Setting it to zero is equivalent to it being melee. The projectile life determines how long the projectile can travel (in terms of seconds) before it self-destructs. The destroy effect is the effect is the effect of when it collides with anything. Checking the Is Enemy Projectile transforms the projectile into the enemy projectile. If not, it sets the projectile as the projectile for the player. The damage is how much damage the projectile causes.

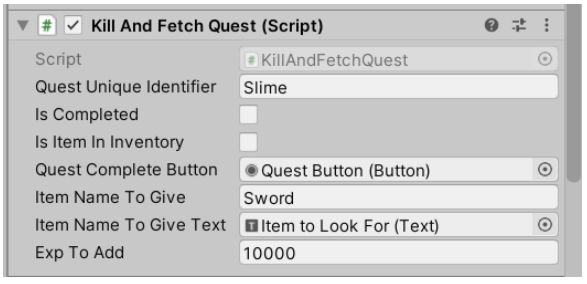
Dialogue & Questing

We have already discussed the Dialogue in Chapter 2, so we can now move on to the Quest Manager.



We can find the Quest Panel in the Dialogue/Quest Giver game object.

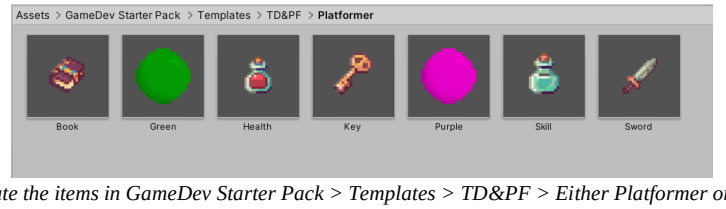
This is the expanded quest panel.



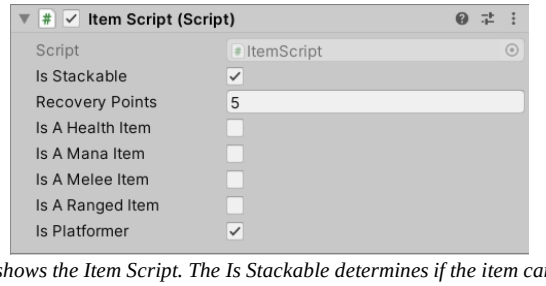
The Quest Unique Identifier is the save name for the quest. If you want to have another quest, make sure that no tow quests have the same identifier. The Is Completed is checked when the quest is completed. Do not touch this. Also, the Is Item In Inventory is checked if the quest requirement is accomplished. The Quest Complete button is the Button (Quest Button in the hierarchy) that changes text based on the current state of the quest. The Item Name To Give is the item name of the item that you want it to be the requirement. In this case, the item is Sword. The item name to give is basically the item that the player needs to find. Lastly, the Exp to Add is how much exp is given to the player after completing the quest.

Items

The items are the ones that the player can pick up in the top-down and platformer games. Let's look at how to edit an item.



You can locate the items in GameDev Starter Pack > Templates > TD&PF > Either Platformer or Top Down.

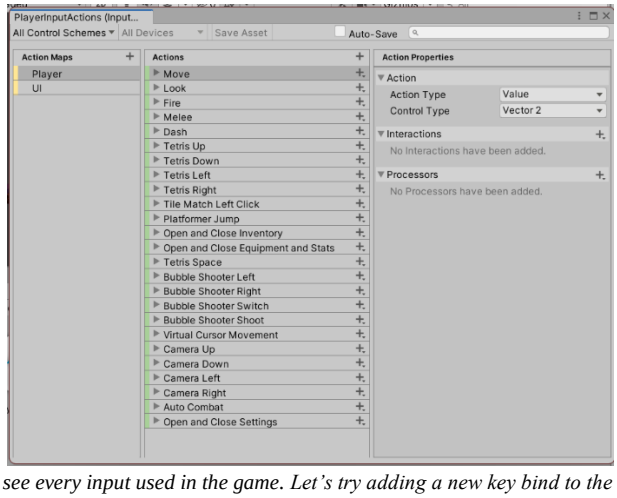


Clicking one of the items shows the Item Script. The Is Stackable determines if the item can be stacked or not in the inventory. The Recovery Points is how much the item can replenish the player (if it is either a health or mana item). The next four items determine the function of the item. You should only pick one or none at all. The naming convention is already intuitive so you can just refer to those. The Is Platformer is checked if the game mode is platformer. The items would then simulate gravity. If not, the items would be suspended in position.

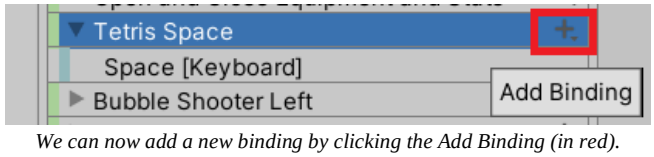
Input Binding

The easy-to-use input binding is provided to the package by using Unity's New Input System. We can easily add or change input bindings for multiple controls. Let's try adding a new key bind to a control.

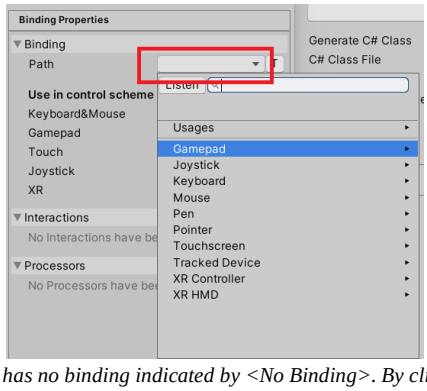
We can search for the Inputs manager in the Asset window, which is named PlayerInputActions with a folded paper and blue bolt icon. We can double click on it to open the Input Manager.



We can now see every input used in the game. Let's try adding a new key bind to the Tetris Space.

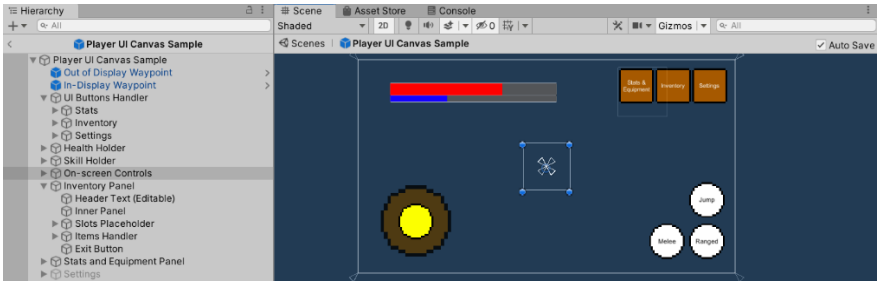


We can now add a new binding by clicking the Add Binding (in red).



It will show below Space that is has no binding indicated by <No Binding>. By clicking the Path (in red), you can then select from any type of controller the binding that you want. The binding can now be used in the Tetris template.

On-screen Buttons



The On-Screen Buttons

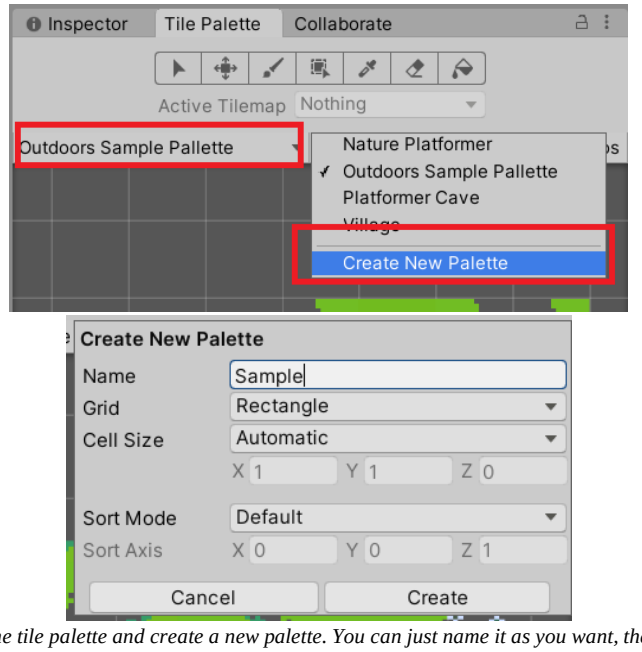
The on-screen buttons can be used for touch-based platform, especially mobile. With the Input System, we can easily add a new screen button with an attached key binding to it. We can see at the example in the Player UI Canvas Sample. We can see the On-screen buttons components attached to it. You can create an on-screen joystick or a simple button. Use the sample in the Player UI Canvas Sample as the reference.

Creating Tilemaps and Palettes

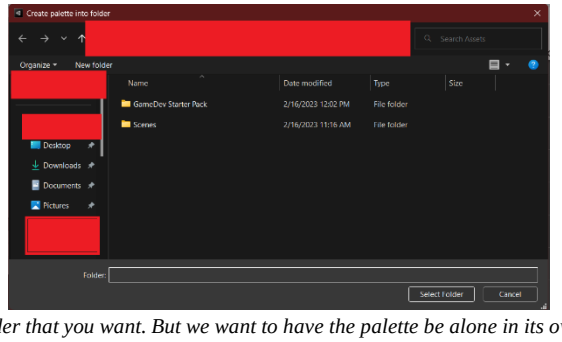
This is how you create a tile palette and tilemaps. First, search for your very own tilemap sprite sheet or you can easily use the ones provided in the package. You can skip to the tilemap painting if you want to use the GDSP tile palettes.

The internet has many resources for tilemap samples. We will use this one as example. To import an image in Unity, just drag and drop the image from the File Explorer to the Asset window in Unity.

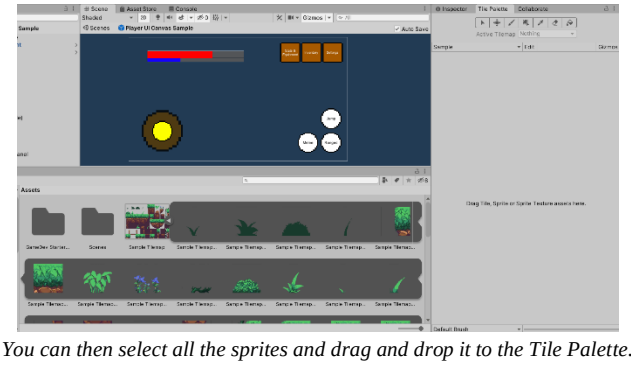
Like the sprite sheets, we will change the Sprite Mode to Multiple.



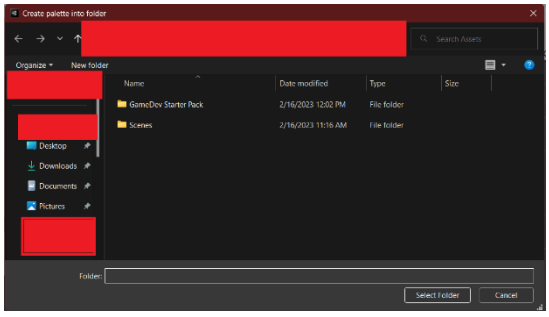
We can then open the tile palette and create a new palette. You can just name it as you want, then click create. You can then follow the steps below:



You can choose any folder that you want. But we want to have the palette be alone in its own folder, so we can go ahead and create a new folder. We can then select that folder, and the tile palette will be saved there.

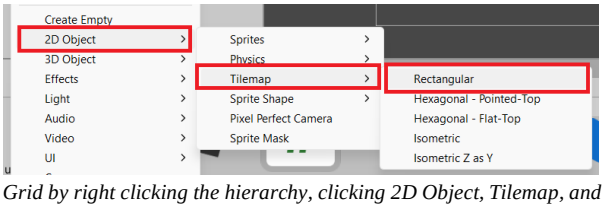


You can then select all the sprites and drag and drop it to the Tile Palette.

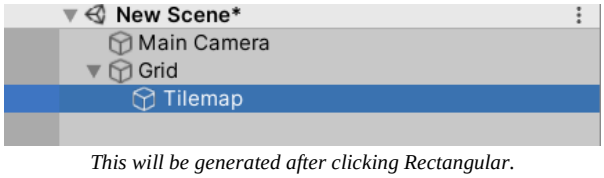


This will show again but reselect the folder that you previously selected.

You now have your own tile palette. We will now start to paint the map. You can easily do this with the templates, but to show the step-by-step process, we will start in an empty scene.

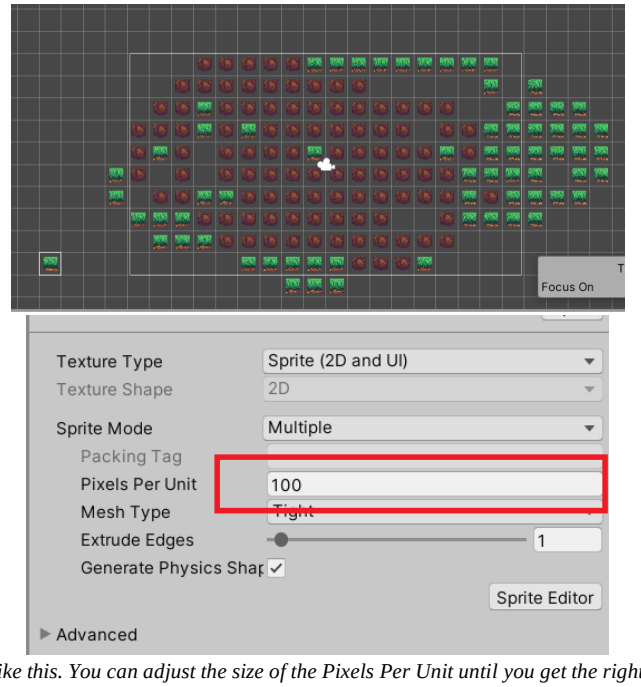


We can create our Grid by right clicking the hierarchy, clicking 2D Object, Tilemap, and then Rectangular.

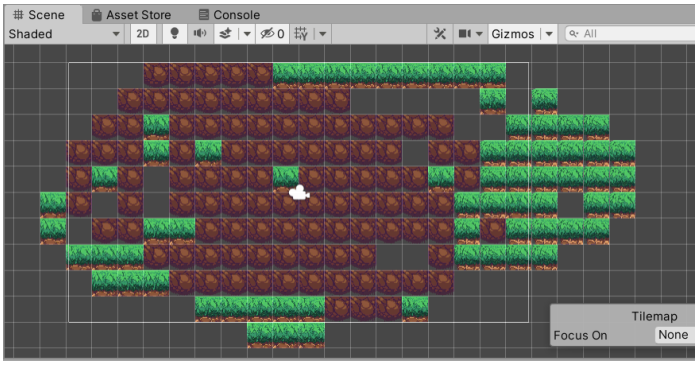


This will be generated after clicking Rectangular.

The default brush is pre-selected in the Tile Palette. You can now then design the whole map.



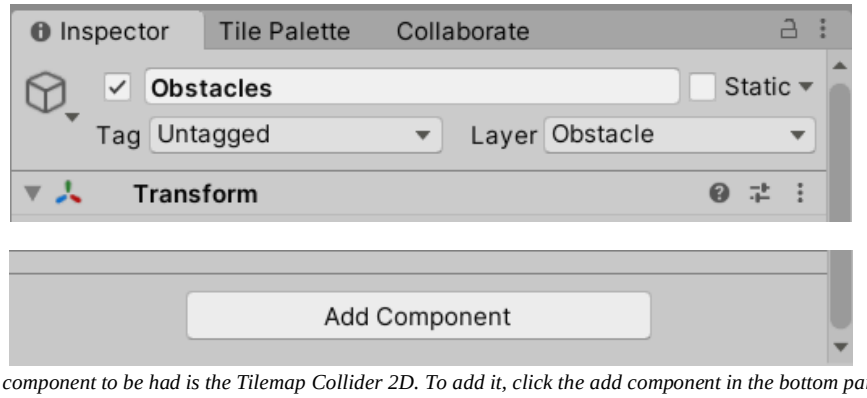
This would look like this. You can adjust the size of the Pixels Per Unit until you get the right size. It will look something like the image below.



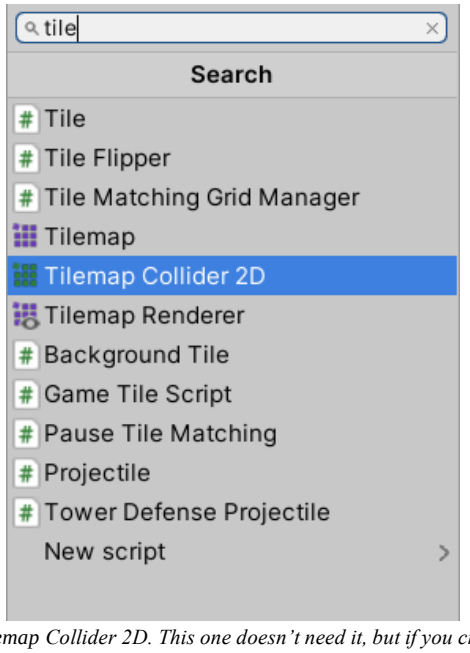
Adding Obstacles for AI

We can use the Grid to create obstacles for AI. We can see an example in the Top-down template.

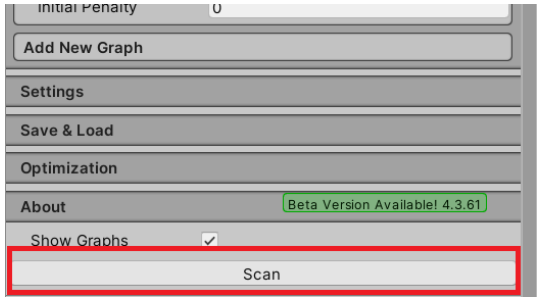
We already have an obstacle layer in this template. We can see that in the image below, the Layer is set to Obstacle.



Another component to be had is the Tilemap Collider 2D. To add it, click the add component in the bottom part of the Inspector.



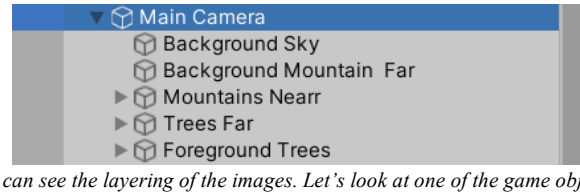
You can then search for the Tilemap Collider 2D. This one doesn't need it, but if you create a new obstacle layer, you need to do this.



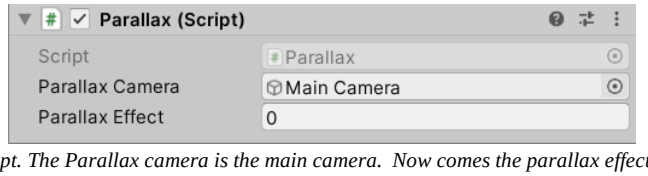
We can then go to the AI Manager and hit Scan. Now, the obstacles are generated, and the AI will avoid that when moving around the map.

Parallax

The Parallax is an exclusive function in the Platformer. It gives the illusion that the player is moving on a background with depth, with layers moving in different speed, the closest moving the fastest, and the farthest moving the slowest. Let's look at the children of the main camera on the Platformer template.



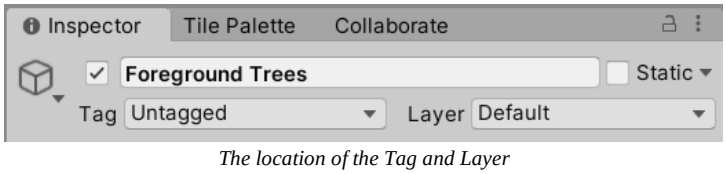
We can see the layering of the images. Let's look at one of the game objects.



This is parallax script. The Parallax camera is the main camera. Now comes the parallax effect. The scaling of the effect is from 0 to 1. 0 being the fastest to move, and 1 doesn't move at all. If you want your foreground to move as fast as you, you can set it to zero.

Tags and Layers

This is the last segment of the package manual. The Tags and Layers can be found in the top of the Inspector. Every tag and layer are different per game object. Since GDSP is set up to be plug and play, setting the tags will not be necessary. This is the location of the Tags and Layers.



The location of the Tag and Layer