Lets Talk Unity

Ashley and Griffin
(you can tell who chose this vibe)

ENTER

01

02

03

04

The Plans

Coding Basics

Got to know this before moving on to more

Live Demonstration

Making our first 3d game

Intro to Space Invaders

Working with the assets

01.
Coding Basics

Unity is made in C# quite similar to Java

01
02
03
04

What is C#?

- C# is an object-oriented language developed by Microsoft
- **Object-oriented**: Code is organized into blocks called "objects" (classes) that have their own data and functionality
    - Helpful for separating game objects so that they don't interfere
    - Called Object-Oriented Programming (OOP)
- Many of the concepts from other programming languages (Java, Scratch, etc.) transfer into C#

Variables

- Variables are stored values in the computer that we can change and compare to alter our code's functionality
- Every variable has two parts:
    - A **type**, which determines what kind of data it can store,
    - A **value**, which is the actual content of the variable
- In OOP, variables can also have **access modifiers**, which determines where their data can be accessed from
- In C#, we create a new variable like this:
    - `<access modifier> <type> variableName = <value>;`

Variable Types

- Let's talk about some basic variable types!
- `int` (short for integer) - stores whole number values
  - `private int points = 0;`
- `float` (floating point number) - stores decimal numbers
  - `private float position = 5.4F;`
- `double` (double precision float) - float with more information
  - `public double x_position = 3.24534D;`
- `string` - a series of letters and numbers
  - `public string myName = "Griffin";`
- `bool` (short for boolean) - a number that can be true or false
  - `bool gameOver = false;`

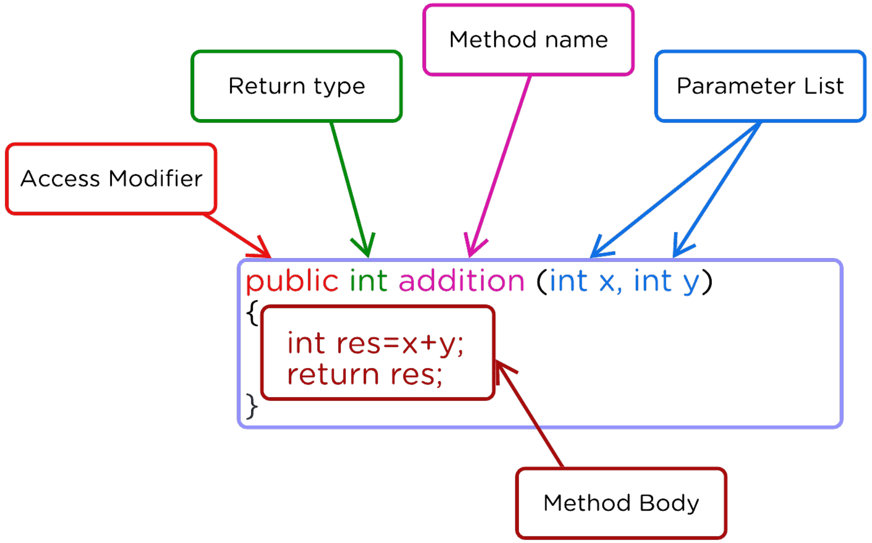
Let's practice declaring variables!

- How would I create a variable for the user's favorite color?
- Fill in the blanks:
  - `public <type> favoriteFood = "pizza";`
  - `public <type> pi = <value>;`
  - `public <type> itsRaining = false;`

Methods

- A group of code instructions that take in some value and produce output
- Part of a class
- Values given to functions are known as **arguments**
    - Arguments look like variables (have a type and a name)
- Values produce their output either by executing code or giving an explicit **return value**
    - Return values also have a type
- Methods can also have **access modifiers**, but we won't really deal with that now

Method Example



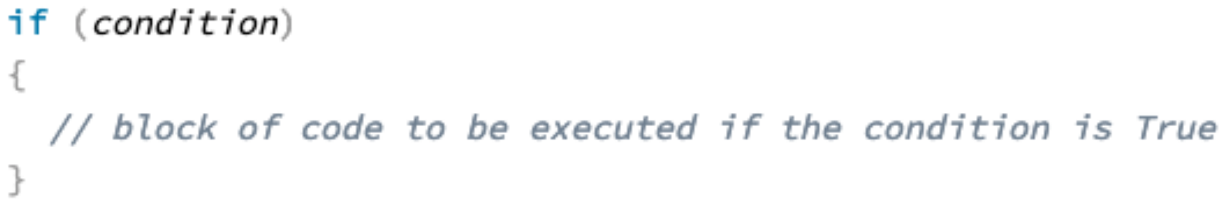
```
public int addition (int x, int y)
{
  int res=x+y;
  return res;
}
```

01
02
03
04

01
02
03
04

Control Flow

- So we have some variables that have values…
- How do we control how our code executes?
- Most basic is an **if statement** which runs code only if a certain condition is met
    - Your condition has a bool type because it can only be true or false



```
if (condition)
{
  // block of code to be executed if the condition is True
}
```

If statement examples

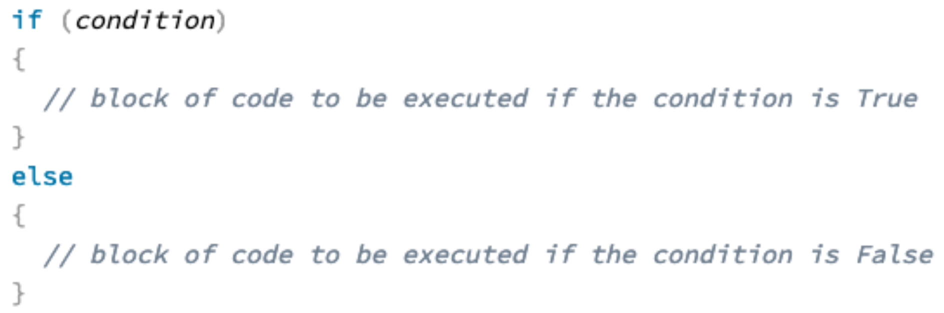
```
if (20 > 18)
{
  Console.WriteLine("20 is greater than 18");
}
```

```
int x = 20;
int y = 18;
if (x > y)
{
  Console.WriteLine("x is greater than y");
}
```

01
02
03
04

If-else statement

- Sometimes we only want to run code if a statement is **not** true
- We could do this outright (create a boolean that has the opposite value of what we're looking for) or…
- We could use an if-else statement!
- If the condition is true, it runs the **if** portion, otherwise it runs the **else** portion



```
if (condition)
{
  // block of code to be executed if the condition is True
}
else
{
  // block of code to be executed if the condition is False
}
```

If-else statement example

```
int time = 20;
if (time < 18)
{
  Console.WriteLine("Good day.");
}
else
{
  Console.WriteLine("Good evening.");
}
// Outputs "Good evening."
```

Doing things over and over again...

- What if we wanted to run a certain of instructions multiple times?
- Once again, we could do this outright by copy-pasting the code we want to run a bunch of times...
    - Does this always work?
- Ponder this example: The user is taking some number of classes that we don't know and we want to get all the names of their classes
    - First we ask them how many classes they are taking and store it in a variable
    - How many times do we copy-paste the code to ask the name of the class...?

Loops

- For this, we use a **loop** which is a way of telling the computer we want to run a
- The easiest type of loop is a **while loop** which is similar to an if statement repeated many times
- The code in a defined section will run as long as a certain boolean value or condition is true

```
while (condition)
{
  // code block to be executed
}
```

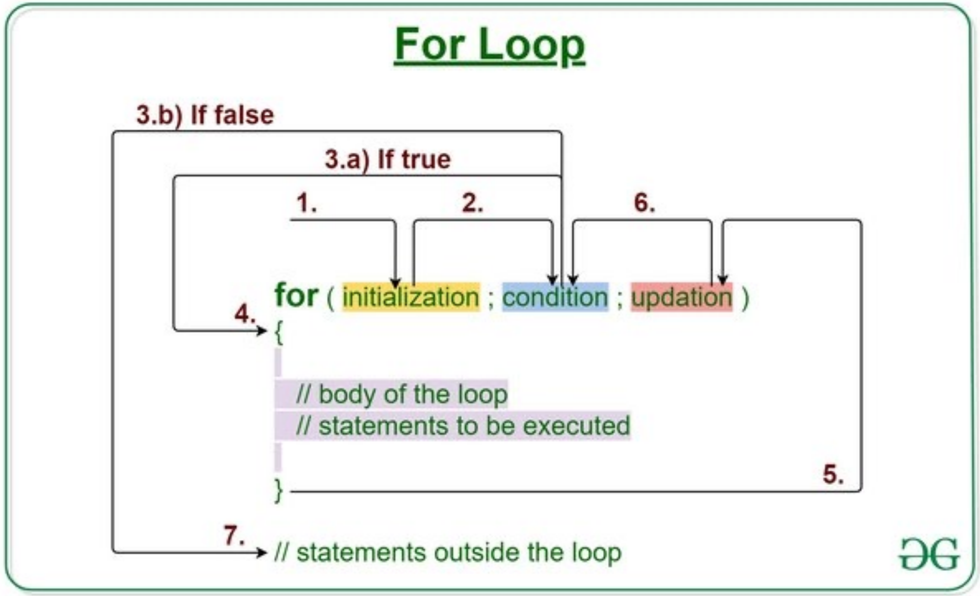
While loop example

```
int i = 0;
while (i < 5)
{
   Console.WriteLine(i);
   i++;
}
```

For loop

- The other type of loop in C# is known as a **for loop** which is makes it easier to write code in exchange for being a bit harder to understand
- For loops create a variable, check a condition, and update
- There are 4 parts to a for loop:
  - **Initializer** - the code that initializes the variable that will be updated during each iteration
  - **Condition** - what we will check about the variable during each iteration
  - **Updater** - the code that changes the variable during each iteration
  - **Code** - the actual code that will run on each iteration

For loop structure



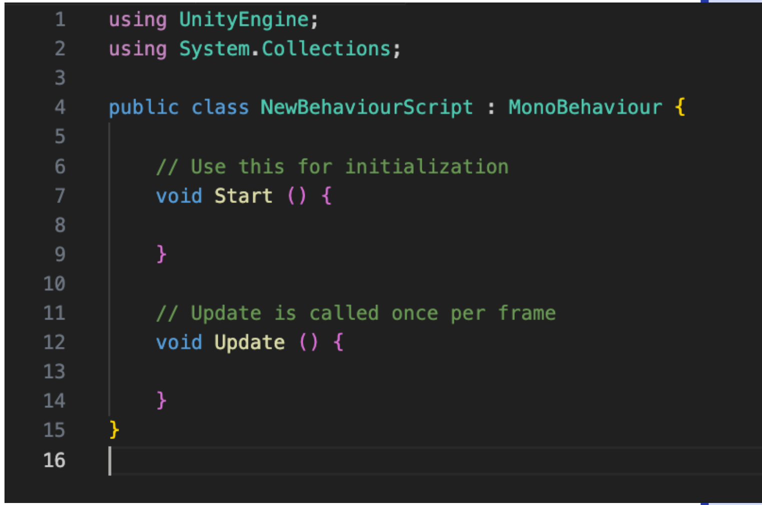
```
for ( initialization ; condition ; updation )
{

  // body of the loop
  // statements to be executed

}
// statements outside the loop
```

For loop example

```
for (int i = 0; i < 5; i++)
{
  Console.WriteLine(i);
}
```

What code looks like in Unity

- This is a boilerplate for a new behavior in Unity named "NewBehaviorScript"
- It has two **methods**, one called "Start" and one called "Update"
- The code in "Start" will be called once when the program starts, "Update" called every frame
- What are the **arguments** of Start?
- What is the **return type** of Start?
- What is the **access modifier** of NewBehaviourScript?

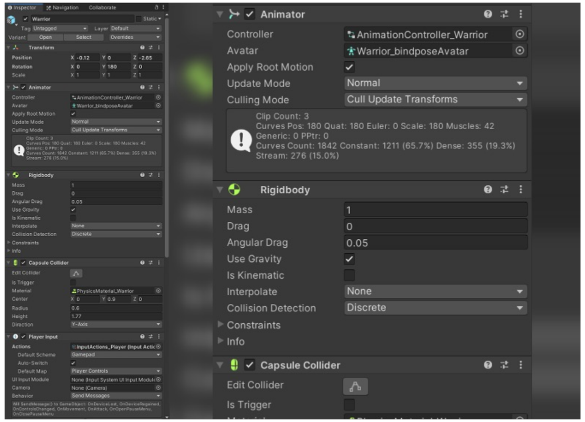


```
using UnityEngine;
using System.Collections;

public class NewBehaviourScript : MonoBehaviour {

    // Use this for initialization
    void Start () {

    }

    // Update is called once per frame
    void Update () {

    }
}
```

01
02
03
04

Remember access modifiers?

- Access modifiers control where we see our variables
- **Private** variables are only visible/changeable from within our code
- **Public** variables are visible/changeable from the Unity editor
    - Helpful for passing information from our game to our code



```
Warrior                         Static
Tag Untagged        Layer Default
Variant  Open  Select  Overrides
Transform
Position   X -0.12   Y 0     Z -2.65
Rotation   X 0       Y 180   Z 0
Scale      X 1       Y 1     Z 1
Animator
Controller          AnimationController_Warrior
Avatar              Warrior_bindposeAvatar
Apply Root Motion   ✓
Update Mode         Normal
Culling Mode        Cull Update Transforms
Clip Count: 3
Curves Pos: 180 Quat: 180 Euler: 0 Scale: 180 Muscles: 42
Generic: 0 PPtr: 0
Curves Count: 1842 Constant: 1211 (65.7%) Dense: 355 (19.3%)
Stream: 276 (15.0%)
Rigidbody
Mass                1
Drag                0
Angular Drag        0.05
Use Gravity         ✓
Is Kinematic
Interpolate         None
Collision Detection Discrete
Constraints
Info
Capsule Collider
Edit Collider
Is Trigger
Material            PhysicsMaterial_Warrior
Center     X 0    Y 0.9   Z 0
Radius              0.6
Height              1.77
Direction           Y-Axis
Player Input
Actions             InputActions_Player (Input Actic
  Default Scheme    Gamepad
  Auto-Switch       ✓
  Default Map       Player Controls
UI Input Module     None (Input System UI Input Modul
Camera              None (Camera)
Behavior            Send Messages
Will SendMessage() to GameObject: OnDeviceLost, OnDeviceRegained,
OnControlsChanged, OnMovement, OnAttack, OnOpenPauseMenu,
OnClosePauseMenu
```

01
02
03
04

02.
Dodgeball

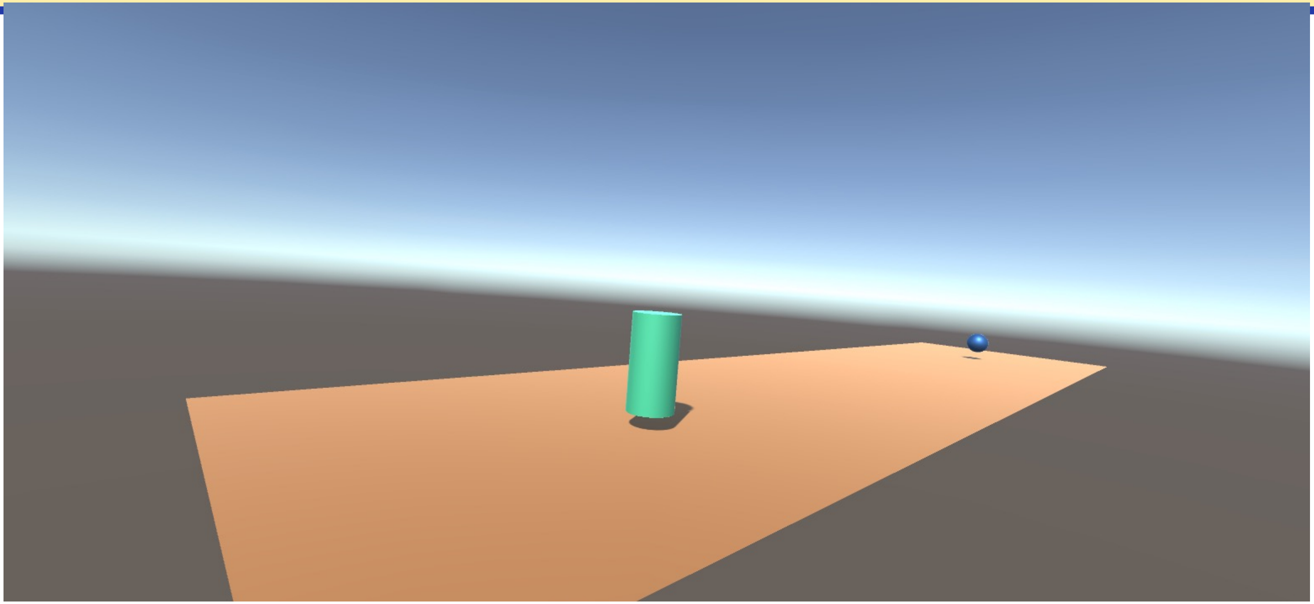
Slides and Demos and Fun(?)
(oh my!)

01

02

03

04



The Goal....kinda

01
02
03
04

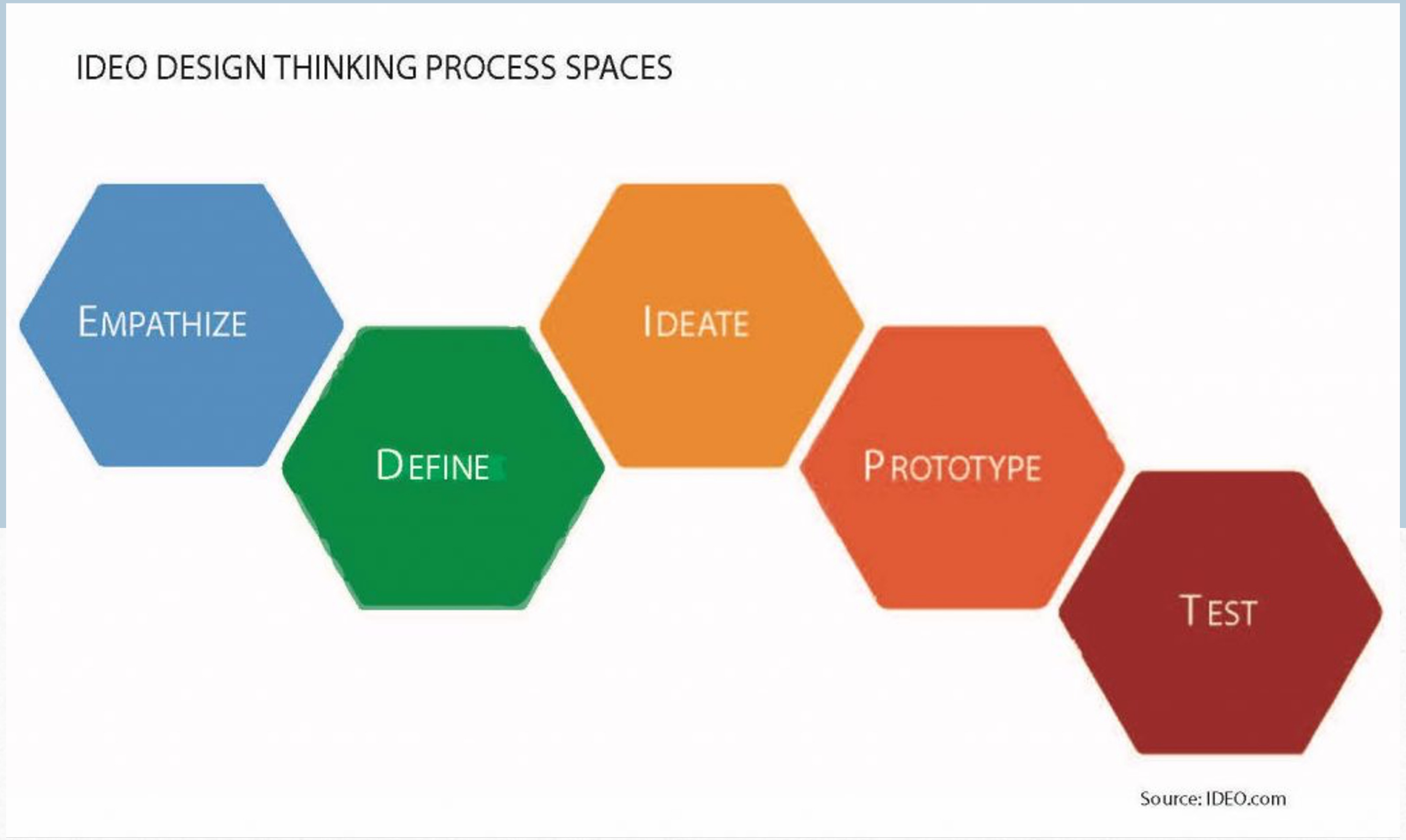
To the Unity Hub!

We will be working all at the same time making this game together. So if you have any questions at all or if we are going too fast stop us

Introduction to Storyboarding

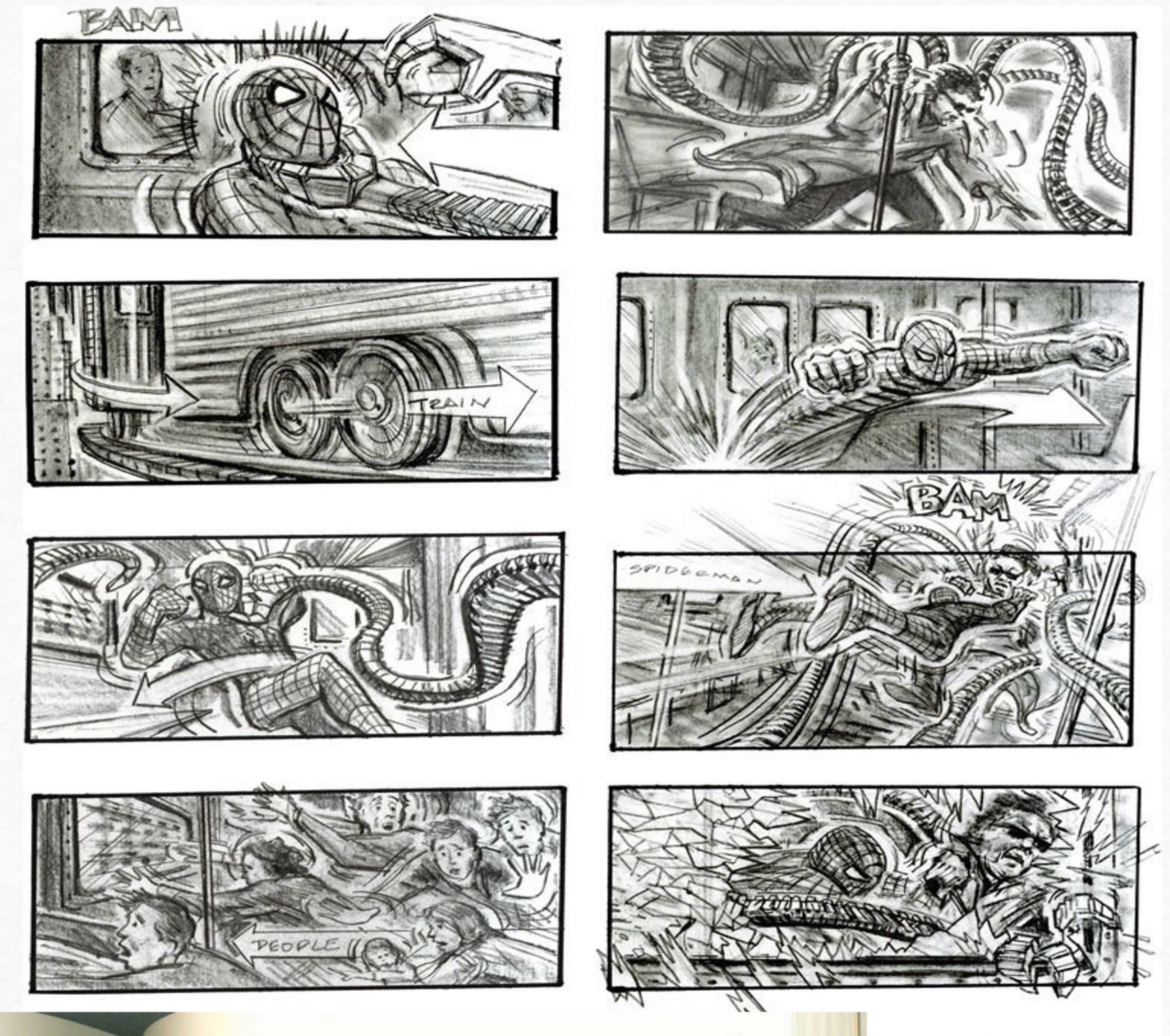
Design Thinking

Design Thinking is a process that all designers use to create and implement creative projects.

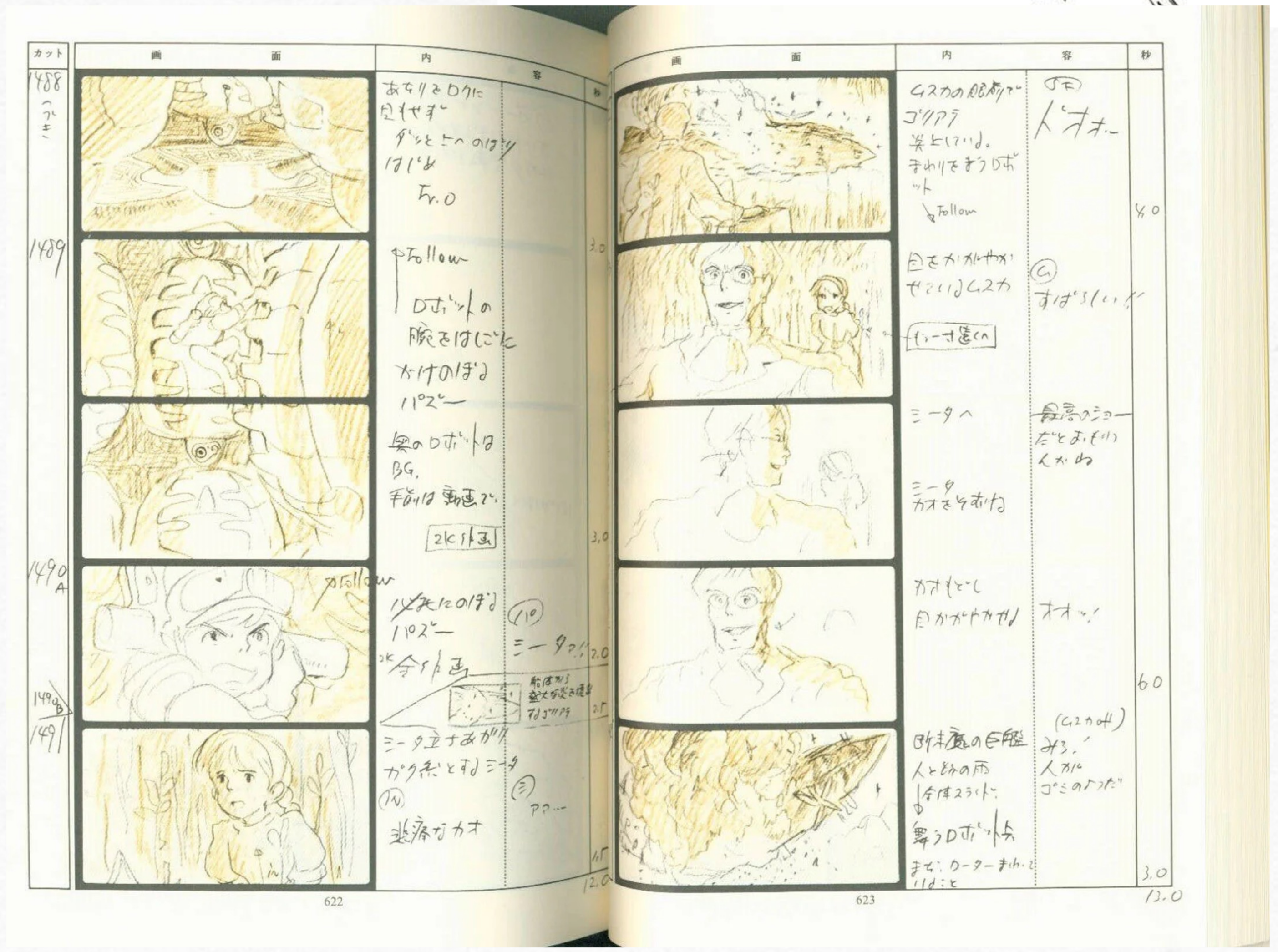


What is a storyboard?

A Storyboard is a **low-fidelity** prototype that serves to help you sequentially plan out a scene. It is a way to create visual notes that aids you in understanding how your story will play out.



YUNI UNI
YUNI THE UNI STUDENT
WASTE
FOOD TRUCK LOCATION
MARKET
FROM WASTE TO TASTE
WASTE
YUNI YUM
BECAUSE IT IS OPEN TO ALL, THOSE IN NEED DOESNT FEEL STIGMATISED



How Storyboarding Helps

VISUALS

- By drawing out your ideas, you have a better picture of the game in your head. The quality of the drawings to a storyboard doesn't matter, the purpose is to provoke visual thought.

PACE

- As you approach the storyboarding process, you must choose what is important to convey and how to convey it. This allows for a natural pacing to form.

PRODUCT

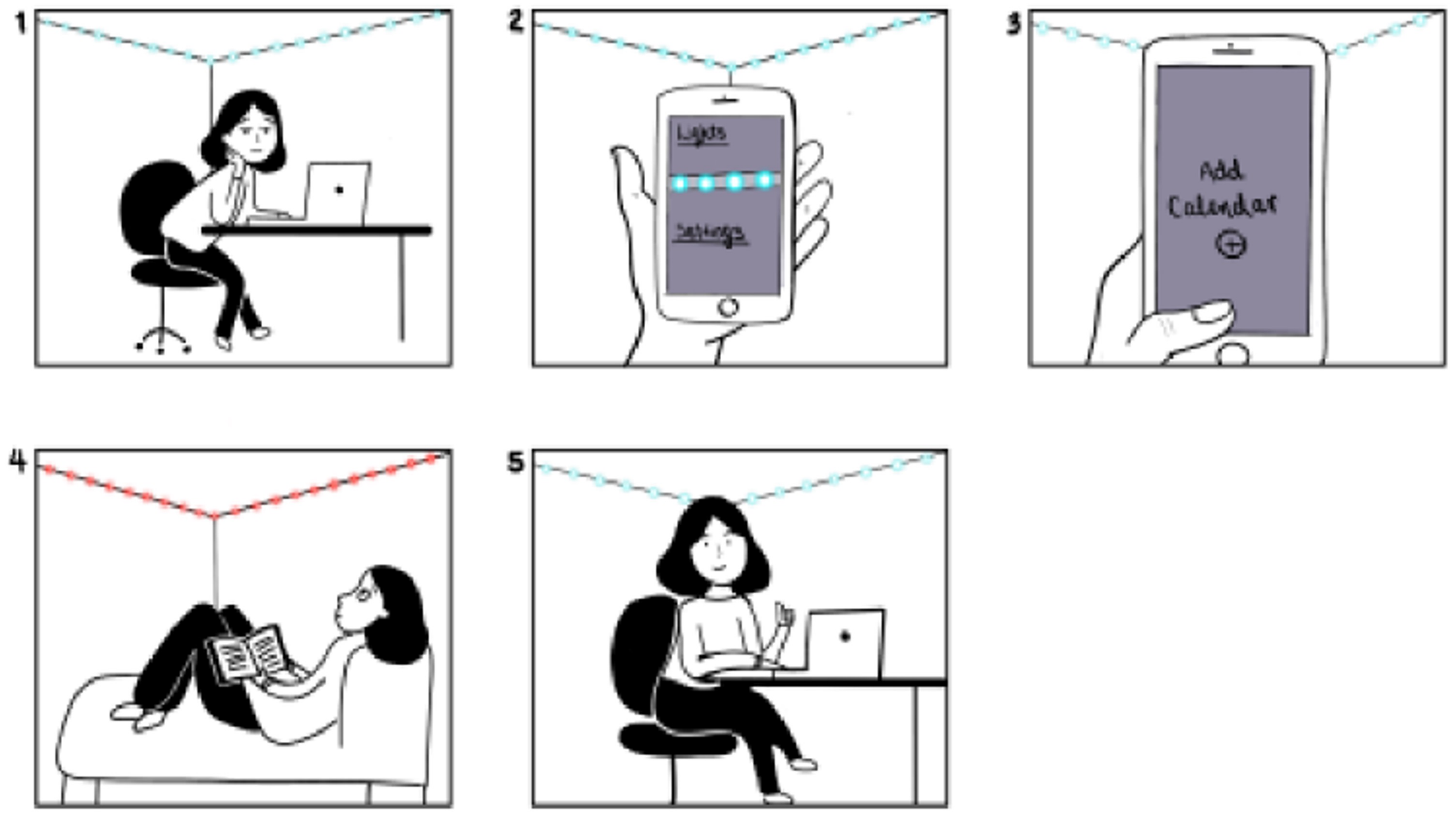
- These storyboards are for you, they are your notes. Creating a storyboard provides you with a physical product that you can look over, refer back to, and edit as you wish.

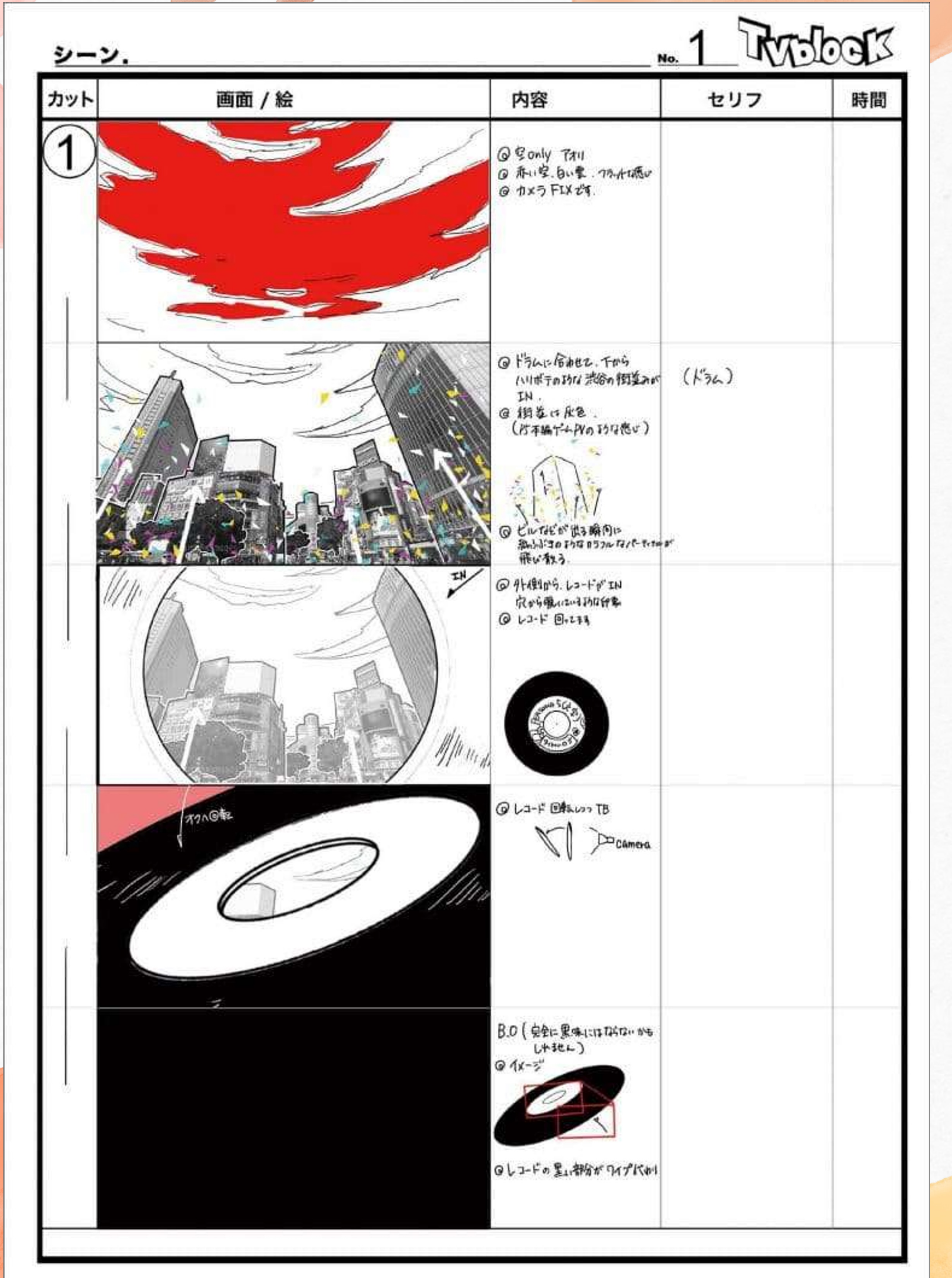
TLDR: Storyboarding can kickstart the creative process and change the way that you imagine the game.

Figma

Figma is a free software used by designers to create storyboards, low and high-fidelity prototypes. It is most widely used for UX/UI Designers for prototyping mobile apps.

Paige's Figma Prototype


1
2
Lights
Settings
3
Add Calendar
4
5



シーン.　No. 1　Tvblock

カット	画面 / 絵	内容	セリフ	時間
1		◎空only アオリ ◎赤い空、白い雲、フラットな感じ ◎カメラFIXです。		
		◎ドラムに合わせて、下から ハリボテのような渋谷の街並みが IN。 ◎街並は灰色。 (既存ゲームPVのような感じ) ◎ビルなどが出る瞬間に 紙ふぶきのようなカラフルなパーティクルが 飛び散る。	(ドラム)	
	IN	◎外側から、レコードがIN 穴から覗いているような印象 ◎レコード 回ってます		
	オクへ回転	◎レコード 回転しつつTB camera		
		B.O (完全に黒味にはならないかもしれません) ◎イメージ ◎レコードの黒い部分がワイプ代わり		

Persona 5 Royal: ATLUS Storyboard

[Clip: (0:37-0:52)]