ゲームを作りましょう

やっと本編です。今回は2Dアクションゲームを作りましょう(簡単なマリオ的な) 今回作るゲームの素材はここでダウンロードしてください。フリー素材 また、今回作るゲームの完成品はこちらです。github

ステージづくり

no3

unityにはシーンという概念があり、場面的なもので例えば1つのステージに1つ のシーンを使ったりします。unityのプロジェクトを初めて開くとシーンが開か れます。1度シーンを作ってみましょう。プロジェクトウィンドウで右クリック してCreate > Sceneを押して下さい



できたら下のプロジェクトウィンドウに新しくシーンが作られます。名前を MainSceneとしましょう。その後MainSceneをダブルクリックするとそのシーンが 開かれます。



また、サンプルシーンは削除してしまいましょう。SampleSceneの上で右クリッ クしてDeleteを選択してください。そのあとに出てくるポップアップもDeleteで 大丈夫です。

素材をプロジェクトに入れます。上でダウンロードしたzipファイルをAssets内 で解凍しておいてください

	(5		
	Create	>	
	Show in Explorer		
	Open		
	Delete		
	Rename		
	Copy Path	Alt+Ctrl+C	
	Open Scene Additive		
	View in Package Manager		
	Import New Asset		
	Import Package	>	
	Export Package		
	Find References In Scene		
	Select Dependencies		
	Refresh	Ctrl+R	
	Reimport		
	Reimport All		
	Extract From Prefab		
	Update UXML Schema		
Accesta > Coopea	Open C# Project		
Assets > Scenes	View in Import Activity Window		
	Properties	Alt+P	
	·		
MainScene SampleSc.			

ここからステージづくりです。2Dアクションのステージ作りにはunityの機能の Tilemapという機能が便利ですので今回はそれを使用します。右上のwindow > 2D > tile Paletteを押してください joken_sampleGame - MainScene - Windows, Mac, Linux - Unity 2021.3.22f1 Personal <DX11>

File Edit Assets GameObject Component Jobs	Window Help		
❸ 増 ▼ ●	Panels	>	
'≡ Hierarchy	Next Window	Ctrl+Tab	
	Previous Window	Ctrl+Shift+Tab	
MainScene	avouts		
↔	Layouts		
	Search	>	
	Unity Version Control		
	Asset Store		
	Package Manager		
	Asset Management	>	
	Text	>	
	TextMeshPro	>	
	General	>	
	Rendering	>	
	Animation	>	
	Audio	>	
	Sequencing	>	
	Analysis	>	
	2D	>	Sprite Editor
	AI	>	Tile Palette
	UI Toolkit	>	
	Visual Scripting	>	

その後開かれたウィンドウでCreate new Paletteを押してNameをなんでもいいの





素材のフォルダ内 Transparent > tilemapからTilemap_packedを選択してくだ さい。その後PexelPerUnitを8にして、インスペクターウィンドウのSplite Mode をMultipleに変更してからSplite Editorを選択してください。出てくるポップ アップはApplyで大丈夫です。

		5 Q Layers V Layout V
	Inspector	a :
€ -	🚛 Tilemap_packed (Text	ure 2D) Import Settings 🛛 🛛 🛛 🕫 🛨 🗄
	tal tran	Open
	Texture Type 🥢	Corite (2D and UI) 👻
	Texture Shape	2D 👻
	Sprite Mode	Multiple 🗸 🗸
	Packing Tag	
	Pixels Per Unit	
	Mesh Type	Tight 🔹
	Extrude Edges	• 1
	Generate Physics Shape	×
		Sprite Editor
	▼ Advanced	
	sRGB (Color Texture)	~
	Alpha Source	Input Texture Alpha 🔹 👻
	Alpha Is Transparency	 Image: A set of the set of the
	Ignore PNG file gamma	
	Read/Write	
	Generate Mip Maps	
	Wrap Mode	Clamp 👻
	Filter Mode	Bilinear 🔹
		- • 1
	Default	
	Max Size	2048 🗸
3 8	Resize Algorithm	Mitchell 🔍 👻
1 25	Format	Automatic
	Compression	Normal Quality
	Use Crunch Compression	

出てきたウィンドウの左上Sliceを選択して画像の通りに変更して、Sliceを押し

Sprite Editor								
Sprite Editor 🔻	Slice 🖥	r Trim						
	Туре			G	rid By Cell	Size		•
	Pixel S	Size		Х	8		8	
	Offset			Х	0		0	
	Paddi	ng		х	0		0	
	Keep I	Empty Re	ects	C				
	Pivot			С	enter			
	Custom Pivot						_	
	Method			Delete Existing 🔹				
	▲	The Delet current Sponce you the Sprite Smart or S	e Existin prites an select A referen Safe slic	g s id r .pp ce: ing	licing method ecreate ther ly. This opera s to get lost. methods ins	i will (n from tion Cons tead.	destroy th m scratch could cau ider using	ie , se
						Slice		

てください。終わったらこのウィンドウは閉じて大丈夫です。

これでTilemap用のスプライトができました。このスプライトをTile Paletteに 入れて下さい。ドラッグアンドドロップで大丈夫です。



次はヒエラルキーで右クリックから、2D object > Tilemap > Rectangular を 選択してください。

	Cut							
	Сору							
	Paste							
	Paste As Child							
	Rename							
	Duplicate							
	Delete							
	Select Children							
	Set as Default Parent							
	Create Empty							
	2D Object	>	Sprites	>				
	3D Object	>	Physics	>				
	Effects	>	Tilemap	>	Rectang	ular		
	Light	>	Sprite Shape	>	Hexago	nal - Pointed-T	ор	
erials	Audio	>	Pixel Perfect Cam	era	Hexago	nal - Flat-Top		
els	Video	>	Sprite Mask		Isometri	c		
105	UI	>			lsometri	c Z as Y		
	UI Toolkit	>	S S S S AAA					
_pico-8-	Camera		ap_pa tilemap_pa	tilemap_pa	tilemap_pa	tilemap_pa	tilemap_pa	tile
emap	Visual Scripting Scene Variables							
es	N					I		

これで、シーンにtilemapを設置できます。ヒエラルキーのTilemapを選択して、 Tile paletteの選択モードで赤丸のグリッドを選択してください。

	Hi	era	rch	у										ć	3 :
+	•	٩	€ Al												R
		V Š	¥ N	lain	Scei	ne*									:
			ğ) Ma	in C	ame	era								
			-	ارو ر د دې	u Filor										
				Ψ	liler	nap									
							-								
														_	5
	1	ſile	Pal	lette									:		×
					h	ê,	1			×	۲	Â	•		
				1	/	*	-	T TO	-		-		-		
				ACI	ive	Tile	map		ema	ip			1		
	fi	irst	_ma	pTil	е					•	Edit	Grid	G	izmo	s
															-
						1									
		_						1							
					1										
		2	-			. • •				1		- 19			G.
	P	-	J.				-		7						
			Ŧ				1					-	-	5	3
			1				1	1	-	а.					
	4	D								5				×	-
						••			•••		• •				
												• •			
			T	2			i E		7	i -	101		22		۲.
	H		٠	÷	7			2	Į		17			X	•
+	Į.		1	Ľ	5	4	כ		ſ		9	×		- 1	2
	1														З.
	F) of	a ult	Brin	ela										
		1010	aunt	Brus	311										

次にTile paletteのペンを選択して、シーンウィンドウに書き込んでください。 そのまま画面を押すことで書き込めます。



次はプレイヤーを作ります。ヒエラルキーで右クリックから2D object > Sprites > Square を選択してください。また、そのオブジェクト名はPlayerに してください。



白い四角が作られると思います。これが今回のプレイヤーです。ただの四角だと 面白くないので見た目を変えましょう。ヒエラルキーウィンドウからPlayerを選 択して、インスペクターウィンドウからSpriteを変更しましょう。画像の丸の部 分を押すとプロジェクト内の画像を検索できるので、今回は素材の Tilemap_packed_104という画像を使用しましょう。

		3	Q,	Layers 🔻	Layout	-
	Inspector					a :
) -	Player				Sta	atic 🔻
	Tag Untagged		Layer	Default		•
	🔻 🙏 Transform				0	⊉ :
	Position	x -0.11		-1.47	Z 0	
	Rotation	X 0		0	Z 0	
	Scale 🗠	X 1			Z 1	
	🔻 🛃 🖌 Sprite Renderer				0	# :
	Sprite	ilemap.	packed	i_104		
	Color					P
	Flip Drow Mode	X Y Simple				_
	Mask Interaction	None				
	Sprite Sort Point	Center				-
	Material	Sprites-	Default			\odot
	Additional Settings					
	Sorting Layer	Default				•
	Order in Layer	0				
	Sprites-Default (Mat	terial)				0 i
	Shader Sprites/Defau					Edit
					- 1	X EX
						Select
						<i>d</i> t
1 25						
						Select
					▼ 300	00

なんかいい感じのキャラクターができたと思います。次はこいつに重力と当たり 判定を与えましょう。インスペクターウィンドウからAdd Componentを選択し て、Rigidbody2DとBoxcollider2Dを選択してください。Rigidbody2Dが物理演算 を行うコンポーネントで、Boxcollider2Dが当たり判定を与えるコンポーネント です。このようにオブジェクトにはコンポーネントを付けることでそれぞれに特 徴を与えることができます。

🔻 👇 🛛 Rigidbody 2D		0 ‡	
Body Type	Dynamic		•
Material	None (Physics Material 2D)		
Simulated	~		_
Use Auto Mass			_
Mass	1		
Linear Drag	0		
Angular Drag	0.05		
Gravity Scale	1		
Collision Detection	Discrete		•
Sleeping Mode	Start Awake		•
Interpolate	None		
► Constraints			_
▶ Info			_
🔻 🔳 🗹 Box Collider 2D		0 ‡	:
Edit Collider	ሌ		
Material	None (Physics Material 2D)		
The physics material used by the any Rigidbody2D or global physi	e collider. Overrides ics material.		
Used By Composite Auto Tiling			
Offset	X 0 Y 0		_
Size	X 1 Y 1		_
Edge Radius	0		
▶ Info			_
Sprites-Default (Mat	erial)	6	
Shader Sprites/Defai		- Edi	

ここで一旦ゲームを実行してみましょう。シーンウィンドウの上にある再生ボタ

ンを押してみましょう。これでゲームのプレビューを確認できます。



実行するとPlayerが下に落ちていったと思います。でもこれは理想の動きではあ

りません。おそらく皆さんが求めている挙動は画面にある地面のようなものに立 つ挙動だと思います。当たり判定をうまく機能させないといけません。先ほど PlayerにつけたBoxcollider2Dコンポーネントは当たり判定を付けるものです。 しかしこれはお互いに当たり判定がついているもの同士でしか作用しません。つ まり地面にも当たり判定を与える必要があります。ヒエラルキーからTilemapを 選択してインスペクターウィンドウからTilemapCollider2Dコンポーネントをつ

けましょう。

	inio									
	🐻 🖌 Tilemap 🛛	Renderer					0	ᅷ		
	Sort Order		Bottom	Left						
	Mode		Chunk							
	Detect Chunk Cu	Illing Bounds	Auto							
	Mask Interaction		None							
	Material		Sprite	s-De	efau	lt			0	
	Additional Settin	ngs								
	Sorting Layer		Default						-	
	Order in Layer		U							
	ⅲ 🗹 Tilemap (Collider 2D					0	칶		
	Max Tile Change	Count	1000							
	Material		None (P	hysio	s N	laterial 2D)				
	Is Trigger									
	Used By Effector									
or	hether the collider not.	r is used by a	n attach	ed ef	fect	tor				
	Info									
	Sprites-D	efault (Mat	erial)					6	• •	
	Shader S	prites/Defau								
							100000		0000	
							100			
							8000	isei	888	
							1	(Sei		
							300			

これでもう一度プレビューを確認しましょう。



立てました。次はプレイヤーの移動を作りましょう。今回は操作が単純なので使 用しませんがある程度操作が複雑なゲームを作る際はInputSysyemという機能を 使った方がいいです。

プレイヤーの移動

input system参考

いよいよ皆さんお待ちかねのコーディングです。まず書き込むファイルを作成し ましょう。プロジェクトウィンドウで右クリックからCreate > FolderでScript フォルダーを作りましょう(任意)

	# Scene	e 😎 Game				
1						
	*			Folder		
	- €			C# Script 2D Visual Scripting	>	
				Shader	>	
		Create	>	Shader Variant Collection		
		Show in Explorer		Testing	>	
		Open		Playables	>	
		Delete		Assembly Definition		
		Rename		Assembly Definition Reference		
		Copy Path	Alt+Ctrl+C	Text	>	
		Open Scene Additive		TextMeshPro	>	
		View in Package Manager		Scene		
		Import New Asset		Scene Template		
		Import Package	>	Scene Template From Scene		
		Export Package		Scene Template Pipeline		
		Find References In Scene		Pretab		
		Select Dependencies		Pretab Variant		
		Refresh	Ctrl+R	Audio Mixer		
		Reimport		Material		
		Reimport All		Lens Flare		
		Extract From Prefab		Lightmap Parameters		٩
		Update UXML Schema		Lighting Settings		
		Open C# Project		Custom Render Texture		
		View in Import Activity Window		Animator Controller		
	-	Descention	Alt- D	Animation		
es		Properties	Alt+P	Animator Override Controller		
				Avatar Mask		
				Timeline		
				Signal		
				Physic Material		
				GUI Skin		
				Custom Font		
				Legacy	>	
				UI Toolkit	>	
				Search		
				Brush		
) to ==	brash		

Scriptフォルダーを開いた状態でプロジェクトウィンドウ内で右クリックから Create > C#scriptを選択し名前をPlayerControllerとしましょう 5

◪▾▯▾▯ਝ▾▦▾▥▾	_			
=		Folder		
▼ -{}+		C# Script		
ġ.		2D	>	
Create	>	Visual Scripting	>	
Show in Explorer		Shader	>	
🕀 Open		Shader Variant Collection		
Delete		Testing	>	
Rename		Playables	>	
Copy Path	Alt+Ctrl+C	Assembly Definition		
Open Scene Additive		Assembly Definition Reference		
View in Deskage Manager		Text	>	
		TextMeshPro	>	
Import New Asset		6		
Import Package	>	Scene		
Export Package		Scene Template		
Find References In Scene		Scene Template From Scene		
Select Dependencies		Scene Template Pipeline		
Refresh	Ctrl+R	Prefab Variant		
Reimport				
Reimport All		Audio Mixer	_	
		Material		
Extract From Prefab		Lens Flare		
Update UXML Schema		Render Texture		
Open C# Project		Lightmap Parameters		
View in Import Activity Window		Lighting Settings		
		Custom Render Texture		
Properties	Alt+P	Animator Controller		
		Animation		
		Animator Override Controller		



ファイルを作成できたらそれをダブルクリックしましょう。次のような画面が開 かれると思います。



とりあえず今必要なのは丸の部分です。それ以外は呪文だと思ってもらって大丈 夫です。画像の内容を簡単に説明すると、上側のStart関数はゲームを実行か、 このオブジェクトが初めて有効化されたときに実行される関数です。例えばキャ ラクターのHPを設定したり初期位置の設定をしたりします。今回はPlayerの初期 位置を設定しましょう。

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
public class PlayerController : MonoBehaviour
{
    // Start is called before the first frame update
    void Start()
    {
       this.gameObject.transform.position = new Vector2(-9, -2);
    }
```

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

}

このように書き込んでみましょう。意味としてはこのオブジェクトの位置を 座標(-9,-2)に変更するというものです。このtransformという部分がオブジェク トの位置とか角度を変更するためのコンポーネントです。これの値をいじるとい う指示を今、スクリプトを通じで書きました。

これを実際のゲームに反映させましょう。このままではPlayerにこの変更は反映 されないので、このファイルをPlayerにコンポーネントとして追加しましょう。 PlayerのインスペクターウィンドウにPlayerControllerファイルをドラッグアン ドドロップで追加できます。



これができたらまたプレビューを確認しましょう。上の再生ボタンを押してくだ さい。



プレイヤーの位置がなんかいい感じのところになったと思います。次はプレイヤ ーの移動を作りましょう。Update関数を書き換えます。PlayerControllerに戻っ てください。次はUpdate関数を変更します。これはゲームを起動している間は常 に繰り返される関数です。一応考え方としては右矢印を押されている間は右方向 に力を加え、左矢印を押されている間は左方向に力を加えるという感じです。た

```
no3
```

```
だし、今回は諸藩の事情によりUpdate関数ではなくFixedUpdate関数を使用しま
す。
```

```
void FixedUpdate()
{
    if(Input.GetAxisRaw("Horizontal") > 0)//もし右矢印が押されていたら
    {
        //右に力を加える処理
    } else if(Input.GetAxisRaw("Horizontal") < 0)//もし左矢印が押されていたら
    {
        //左に力を加える処理
    }
}</pre>
```

このコードを説明すると、Input.GetAxisRawはあらかじめ設定された2つのキ ー(今回は初期設定)のそれぞれに正負を設定して(今回は左矢印に負、右矢印に 正)として、そのキーを押されたときにHorizontalの値を-1か1に変化させるもの です。ので、今回はその値が0より大きいか小さいかで条件分岐をしています。 力を加える処理を書き込んでいきましょう。力を加える処理といってもどういう ことかよくわからないと思います。今回は先ほどPlayerにつけたRigidbody2Dの 機能を使って物理演算でPleyerの移動を作りましょう。(これ以外のやり方もあ ります)力を加えるにはRigidbody2Dをいじる必要があるのでコード内で Rigidbody2Dを取得する必要があります。Start関数とその上を次のように書き換 えてください

```
Rigidbody2D rb;//Rigidbody2D型の変数
void Start()
{
    rb = GetComponent<Rigidbody2D>();//ここでRigidbidy2DをPlayerオブジェクトから取得
    this.gameObject.transform.position = new Vector2(-9, -2);
}
```

これで、rbという変数にPlayerオブジェクトのRigidbody2Dコンポーネントを 代入することができました。これ以降はrbという変数を用いることでPlayerオブ ジェクトの物理演算を操作することができます。

}

}

次にこのように書き換えてみてください。追加した部分はx軸方向に5か-5の 力を加えるという処理です。プレビューを確認しましょう。 おそらくキャラクターは動いたけどすごい勢いでどっかに行ったと思います。こ のAddForceという処理はオブジェクトに力を加えることはできますがその後は考 えていないので、このコードでは右矢印または左矢印を押している間、その方向 に5+5+5+5+5+5というように力が加わっていきます。これではちょっと長押し しただけで吹っ飛んでしまいます。これを防ぐために一定の速度になったら追加 で力は加わらないように変更しましょう。

```
void FixedUpdate()
{
   if(Input.GetAxisRaw("Horizontal") > 0)//もし右矢印が押されていたら
   {
       if(rb.velocity.x く 7)//もし正方向への速度が7以下なら
       {
          //右に力を加える処理
          rb.AddForce(new Vector2(50, 0));
       }
   } else if(Input.GetAxisRaw("Horizontal") く 0)//もし左矢印が押されていたら
   {
       if( rb.velocity.x > -7)
       {
          //左に力を加える処理
          rb.AddForce(new Vector2(-50, 0));
      }
   }
}
```

このようにrb.velocity.xとすることでそのオブジェクトの速度を取得できま す。今回は最大移動速度を7としています。プレビューを確認しましょう。移動 が速すぎて吹っ飛ぶことはなくなったと思います。しかし慣性が残って少し操作 性が気持ち悪いと思います。次はこれを修正しましょう。要は矢印を何も押して いないときのPlayerの水平方向の速度を0にすればいいわけです。ので次のよう に変更しましょう。

```
void FixedUpdate()
{
    if(Input.GetAxisRaw("Horizontal") > 0)//もし右矢印が押されていたら
    {
        if(rb.velocity.x < 7)//もし正方向への速度が7以下なら
        {
            //右に力を加える処理
            rb.AddForce(new Vector2(50, 0));
    }
}</pre>
```

```
}
} else if(Input.GetAxisRaw("Horizontal") < 0)//もし左矢印が押されていたら
{
    if( rb.velocity.x > -7)
    {
        //左に力を加える処理
        rb.AddForce(new Vector2(-50, 0));
    }
} else if(Input.GetAxisRaw("Horizontal") == 0)//どっちも押されていないなら
{
        rb.velocity = new Vector2(0, rb.velocity.y); //速度を0に設定
}
```

このようにすることで慣性はなくなります。これは左右の矢印が押されてい ないときはオブジェクトの速度を0にする処理です。これで横移動は完成です。 次はジャンプを作りましょう。 今回はスペースキーをジャンプ用のキーとして使用しましょう。処理としてはス ペースキーが押されたことを検知してその時に上方向に力を加えたらいいわけで す。

```
void FixedUpdate()
{
   if(Input.GetAxisRaw("Horizontal") > 0)//もし右矢印が押されていたら
   {
       if(rb.velocity.x < 7)//もし正方向への速度が7以下なら
       {
          //右に力を加える処理
          rb.AddForce(new Vector2(50, 0));
       ł
   } else if(Input.GetAxisRaw("Horizontal") く 0)//もし左矢印が押されていたら
   {
       if( rb.velocity.x > -7)
       {
          //左に力を加える処理
          rb.AddForce(new Vector2(-50, 0));
       }
   } else if(Input.GetAxisRaw("Horizontal") == 0)//どっちも押されていないなら
   {
       rb.velocity = new Vector2(0, rb.velocity.y); //速度を0に設定
   }
   if (Input.GetKey(KeyCode.Space))
   {
       rb.AddForce(new Vector2(0, 4), ForceMode2D.Impulse);
   }
}
```

次のように書き換えてください。最後のif構文がジャンプの部分です。 Input.GetKeyDown(KeyCode.Space)によってSpaceキーが押されたかどうかを検 知しています。その後同様にAddForceを用いて力を加えています。今回はジャン プなので垂直方向(y軸方向)に力を加えるようにしています。 プレビューを確認

して、スペースキーを押してみてください。



いい感じにジャンプができました。でもスペースキーを連打すると空中でなんど もジャンプができてしまいます。これを修正する必要があります。 つまりはPlayerが空中にいる(何にも触れていない)ときにスペースキーを押して もなにも起こらないように変更する必要があります。このようにある事象につい て1か0だったり、そうであるか、そうでないかのような処理を行うときはbool型 の変数を使用します。

```
Rigidbody2D rb;//Rigidbody2D型の変数
bool is_field = false;
```

このようにコードを追加してください。このis_field変数がtrueのとき、地 面にいる、falseのとき、空中にいるとして処理を作りましょう。そのためには 地面に触れたときにtrueに変更し、離れたときにfalseに変更する必要がありま す。

それでは地面に接するか離れたことをどのように判定すればいいでしょうか。こ こでは当たり判定を用いて判定します。PlayerのBoxcollider2DとTilemapの TilemapCollider2Dが触れ合ったかどうかを判定すればいいわけです。

```
private void OnCollisionEnter2D(Collision2D collision)
{
    is_field = true;
}
private void OnCollisionExit2D(Collision2D collision)
{
    is_field = false;
}
```

このコードを追加してください。また新しい関数が出てきました。 OnCollisionEnter2Dは自身(この場合Playerオブジェクト)が持っている当たり 判定が他の当たり判定に触れたときに実行される関数です。ので、今回は他の当 たり判定と触れる=地面に落ちるなのでそのタイミングでis_fieldをtrueに変更 します。次にOnCollisionExit2Dですがこれは先ほどと逆で当たり判定同士が離 れたときに実行される関数です。よってこれはジャンプしたときに呼び出される のでis_fieldをtrueに変更します。

これでis_fieldの変更の処理は終わりです。このis_fieldの値によってジャンプ できるかどうかの処理を作ります。先ほどのジャンプの部分を次のように書き換 えます。

if (Input.GetKeyDown(KeyCode.Space) && is_field == true)
{
 rb.AddForce(new Vector2(0, 7), ForceMode2D.Impulse);
}

この&&は"かつ"という意味です。つまり、ifの条件をスペースキーが押され いるかつ、is_fieldがtrueであるとき、というものに変更しました。これでプレ ビューを確認してみると空中でジャンプはできなくなっていると思います。これ でプレイヤーの移動は完成です。