Unity Playgroundマニュアル 日本語訳

翻訳：tamtam (Community Evangelist / Unity Technologies Japan)

履歴

2019年1月17日　ドラフト版初版

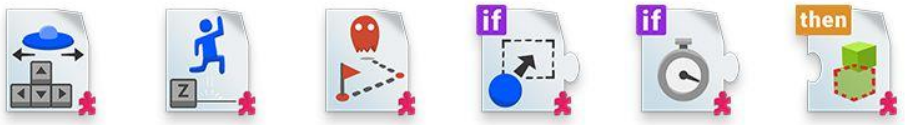
Playground [beta]

Welcome!

あなたが熱狂的なゲーム開発者であろうと、あるいは教育者であろうと、ようこそUnityの世界へ！

Unity Playgroundは、2Dの物理ベースのゲームを作成するためのフレームワークで、コーディングなしでUnity上でゲームを作れるようゲーム開発初心者に教えるにはうってつけです。また、ゲームデザインやレベルデザインを説明するためにも利用することができます。

Unity Playgroundは、使いやすくかつ調整しやすい一連のワンタスクコンポーネント（スクリプト）を提供することで、コーディングの必要性を排除しています。それらワンタスクコンポーネント（スクリプト）を組み合わせることで、複数のゲームジャンルにわたる物理ベースの2Dを作成することができます。



スクリプトのアイコンの例

ぜひ、**Unity Playground**をお楽しみください！

目次

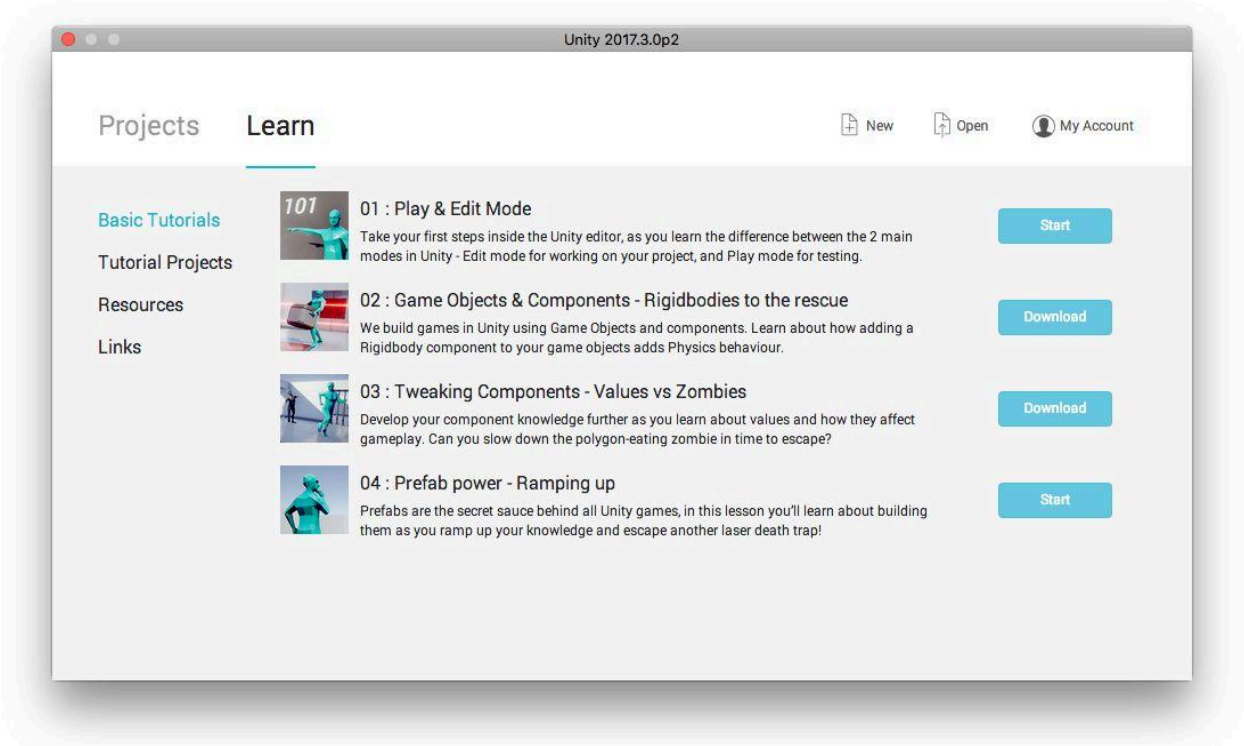
Getting Started（入門）

Unity Playgroundは扱うことがとても簡単であるように意図されています。そして使い始めた人はだいたい30分以内でUnity Playgroundに精通することになるでしょう。プロジェクト構造とUnity Playgroundの実装概念を理解するだけです。それではスクリプトの中身を見たいと思うでしょうから、早速Movementカテゴリに分類されているスクリプトから見ていきましょう。

特定のスクリプトに関する詳細情報が必要な場合は、『リファレンスガイド』を参照してください。unity3d.comのLearnページや、プロジェクトに含まれているPDFファイルなどがあります。

事前準備

Unity Playgroundを使用する前に、Unityの基本的な概念をよく理解しておく必要があります。大変有用なリソースは、Unityエディタ起動ウィンドウやUnityHubで見つけられるLearnタブの中にあるインタラクティブチュートリアルです。



起動ウィンドウの中にある4つのインタラクティブチュートリアル

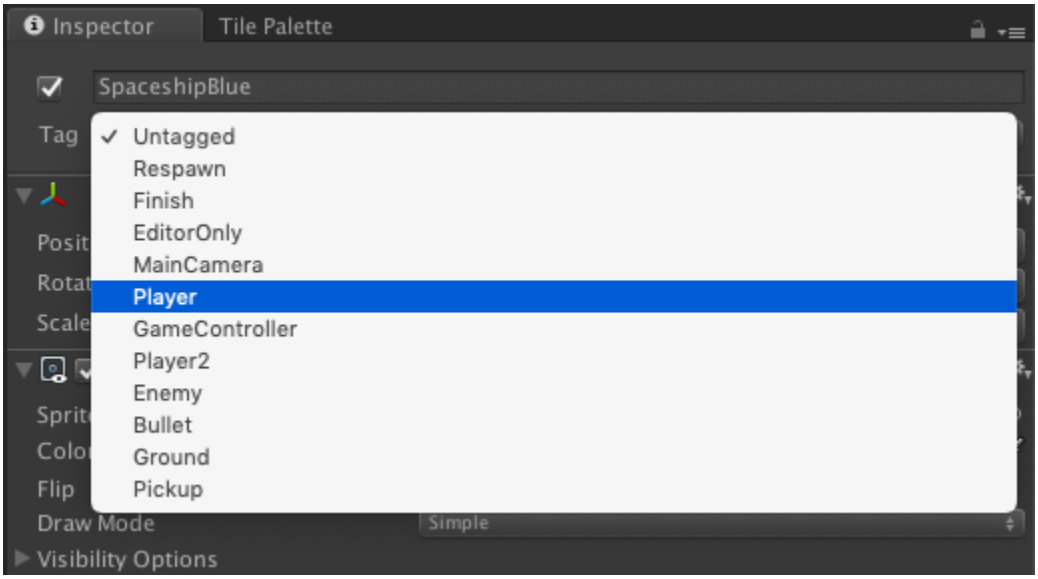
あなたが教育者であり、ワークショップやコースのためにUnity Playgroundを使用しようとしているならば、この後にある【チーティングセクション】で面白い示唆を見つけることができます。

最初のゲーム作り

Unity Playgroundでゲームを作るのはとても簡単です！ まずは簡単なものから作りましょう。

プレイヤーを作成

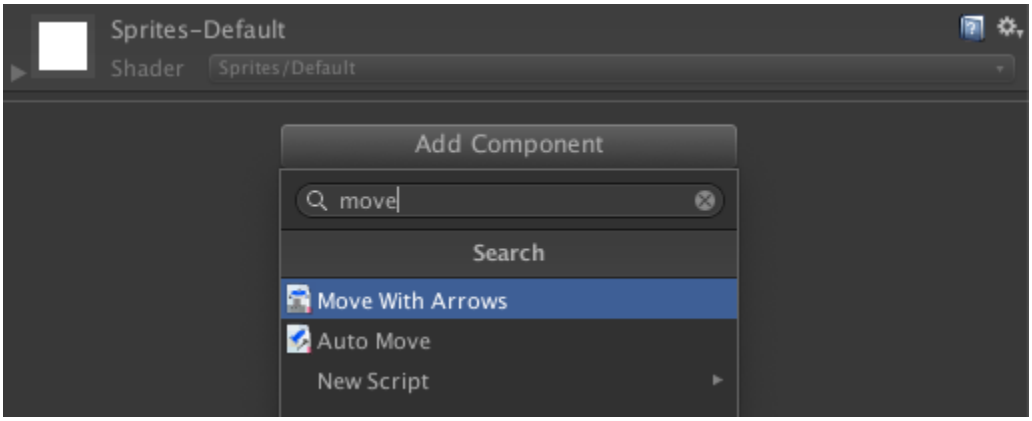
宇宙船のスプライト（「SpaceshipBlue」と「SpaceshipRed」の2種類がありますが、どちらでもOKです）を/ Images / Spaceshipsフォルダから直接シーンビューにドラッグ &ドロップすることから始めます。スプライトをシーンビューにドラッグ &ドロップすると、Unityはそこからgameobjectを作成します。これがプレイヤーになるので、インスペクタウィンドウの上部にあるこのGameObjectに「**Player**」というタグを付けます。



「Player」のタグを選択

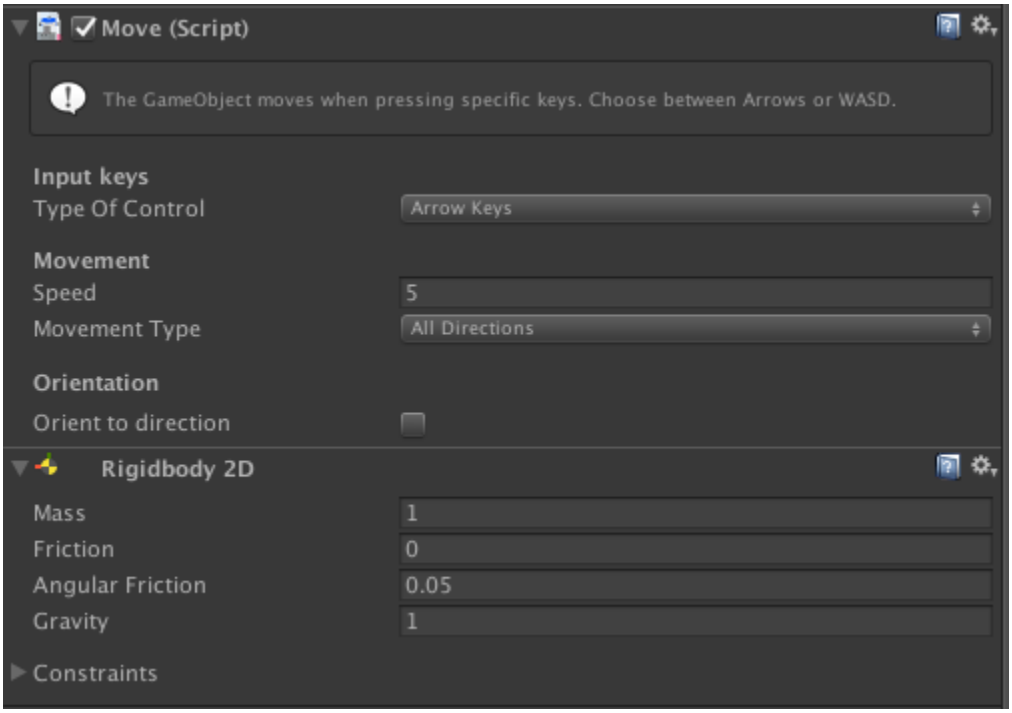
それでは宇宙船を動かしましょう。動かすためには 2つのコンポーネント＝スクリプトが必要です。ユーザー操作に合わせてインタラクティブな動きを提供するための**Move**スクリプトと、宇宙船が物理法則に従うようにするための**Rigidbody2D**コンポーネントです。Moveスクリプトを/ Scripts / Movementフォルダから宇宙船（「SpaceshipBlue」や「SpaceshipRed」）のインスペクタにドラッグ &ドロップしましょう。

もしドラッグ &ドロップができない場合などは、宇宙船のインスペクタの一番下にある「Add Component」ドロップダウンメニューから「move」と入力することでMoveスクリプトを選択できるようになります。Moveスクリプトは最初の結果として表示されます。



インスペクタの「Add Component」をドロップダウン

Moveスクリプトを追加するとすぐにRigidbody2Dコンポーネントも追加されます。これは、Moveスクリプトが機能するためにはRigidbody2Dコンポーネントを必要とするためです。

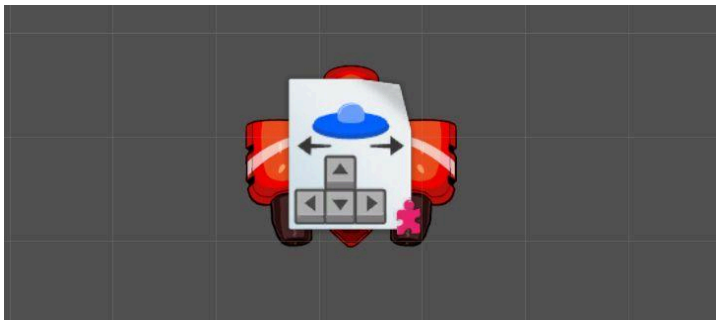


Moveスクリプトをドラッグ &ドロップすると、Rigidbody2Dスクリプトも付いてくる

Note

Scene Gizmos（シーンギズモ）

Unity Playgroundスクリプトのアイコンが大きく、グラフィックが隠れていることがあります。

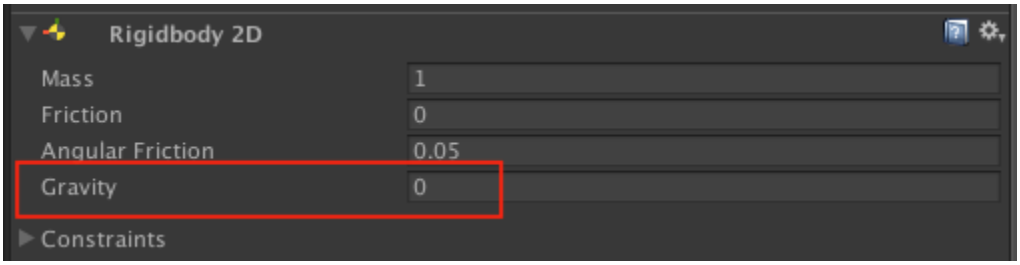


Moveアイコンが宇宙船全体を覆ってしまう

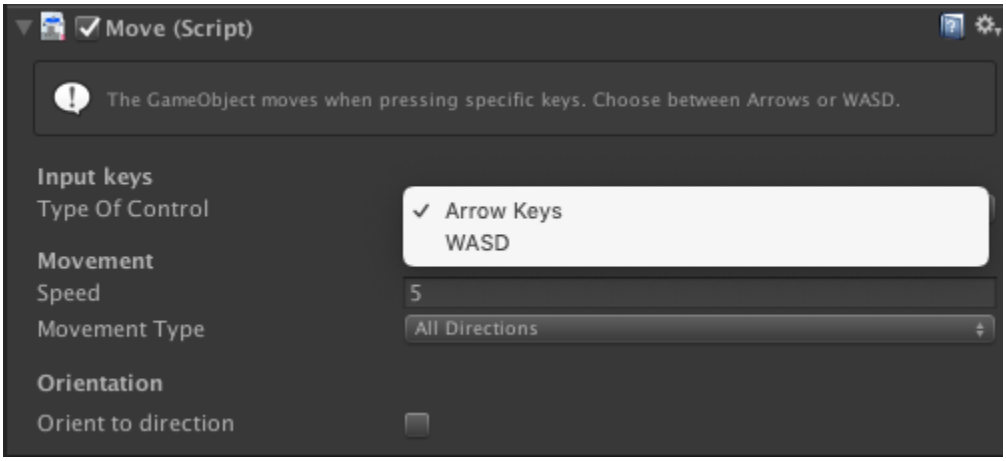
このような場合は、シーンビューのGizmosドロップダウンを使用してサイズを縮小できます。アイコンが正しいサイズになるまで、3Dアイコンスライダを左にドラッグします。

「3D Icons」はシーン内のギズモのサイズを制御します

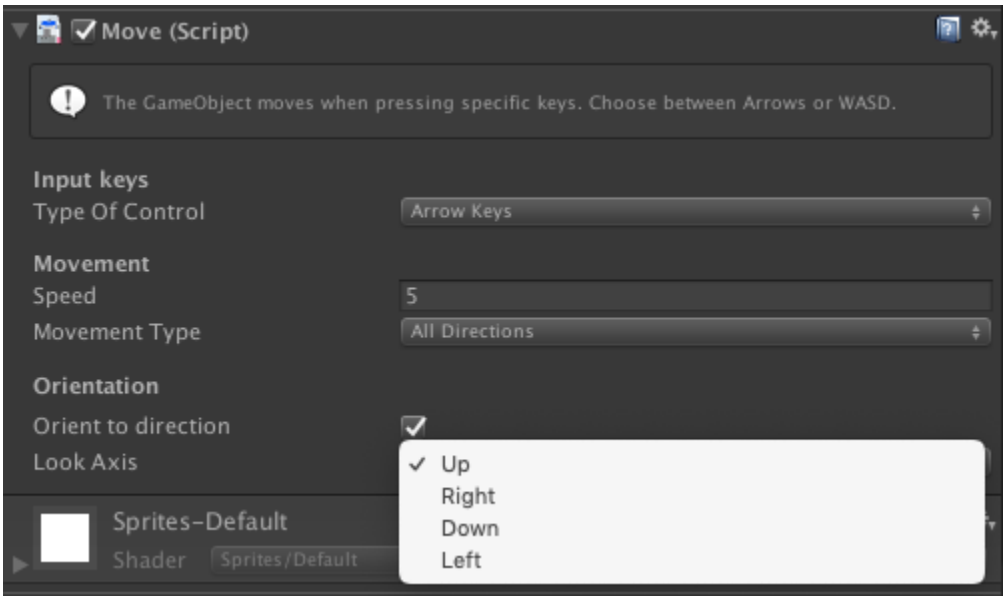
まず、RigidBody2Dコンポーネントの「Gravity」を0に変更しましょう。（このあと行う）プレイボタンを押しても宇宙船は落ちなくなります。



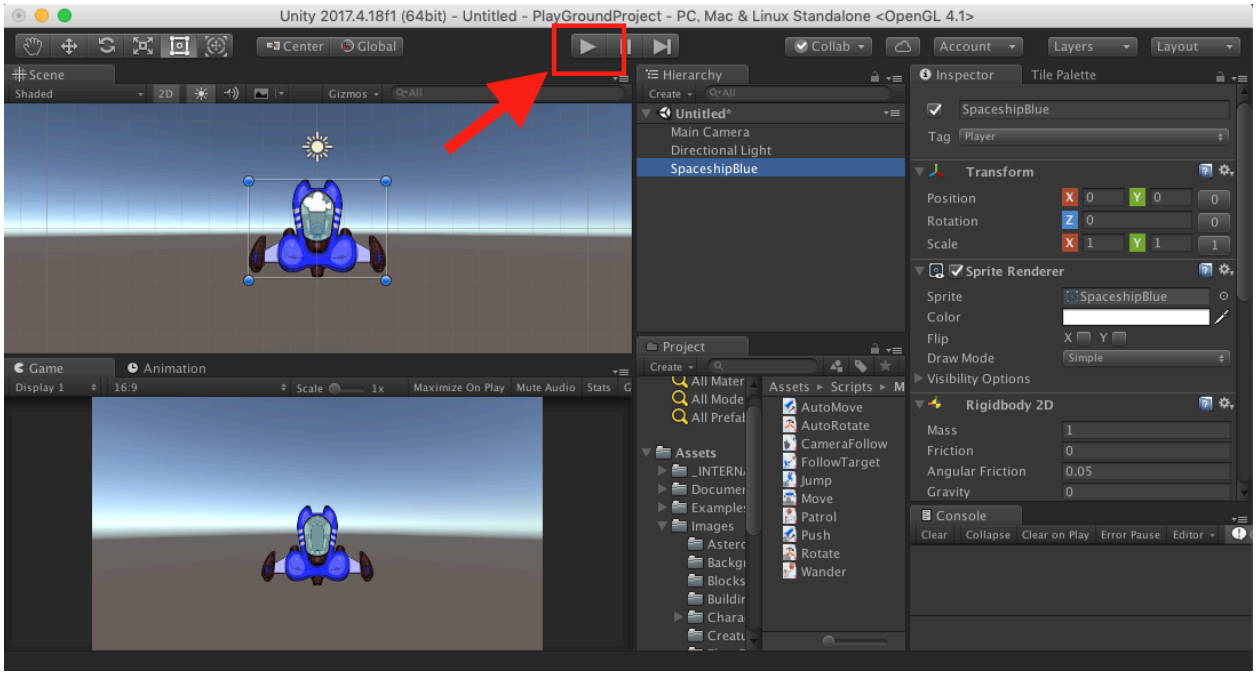
また、Moveスクリプトの「Input keys > Type Of Control」で入力方式を選択できます。Arrow Keysが矢印キーによる入力、WASDはW＝上・A＝左・S＝下・D＝右キーによる入力です。



さらに、宇宙船の向きを入力キーの方向を一致させたい場合は、「Orientation > Orient to direction」にチェックを入れ、「Look Axis」をUpにします。



それではUnityエディタの上部にあるプレイボタンを押して、ゲームをテストしてみましょう。



宇宙船を動かすとものすごい惰性で動いてしまい（＝ドリフトしてしまい）、コントロールすることが難しいと感じるはずです。そこでRigidbody2DのFriction（摩擦）を10に調整します。

Note

プレイモード（Play Mode）

プレイモードでインスペクタにあるスクリプト・コンポーネントの値を編集した場合、ゲームが停止すると変更内容が失われます。プレイモードになっていない場合にのみ変更を反映できることを忘れないでください！　あなたが失うことを気にしない一時的な値をテストしている場合にのみ、プレイモードで変更してもよいです。

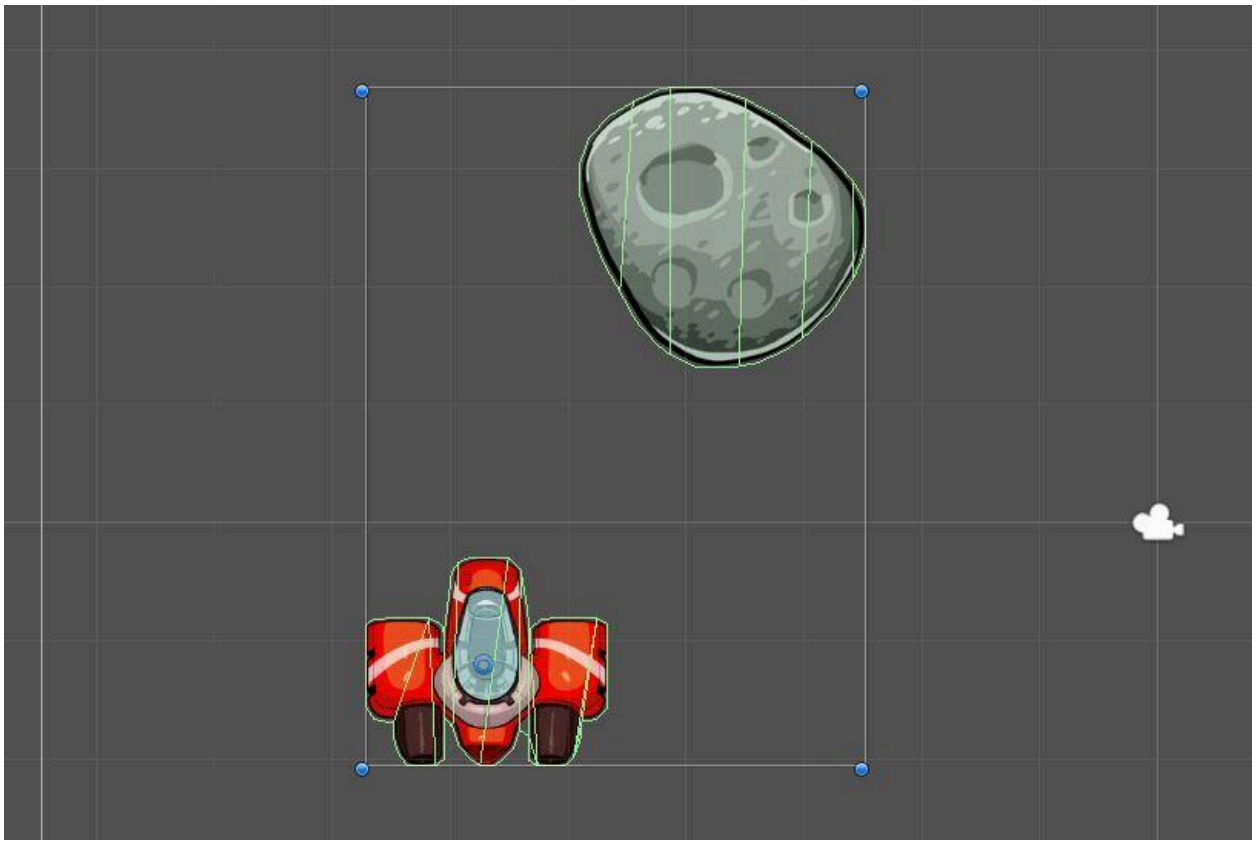
プレイしてみると、惰性（＝ドリフト）は無くなっていることがわかります。また、移動するキーを入力しても宇宙船が大きく動かなくなっていることもわかります。これは、動かしていた力が摩擦によって打ち消されているためです。

なので、Moveスクリプトでも**Speed**パラメータを調整する必要があります。8にして、もう一度プレイボタンを押してみましょう。宇宙船をコントロールするのがもっと簡単になったはずです。これで初めてのゲームプレイの調整をしました。(おめでとうございます。あなたはゲームデザイナーです！)

障害物や衝突処理を追加

この時点でプレイできるものが作れましたが、実際にはゲームではありません。/ Imagesフォルダから小惑星のスプライト(Asteroidsフォルダにあります)をドラッグ &ドロップして宇宙船が避けなければならないいくつかの障害を追加しましょう。この前と同様に、UnityはそこからGameObjectを作成します。

Rigidbody2DとPolygonCollider2Dの2つのコンポーネントを追加する必要があります。PolygonCollider2Dは、このオブジェクトが他のオブジェクトと衝突(接触)できるようにします。このコンポーネントも宇宙船に追加する必要があります。

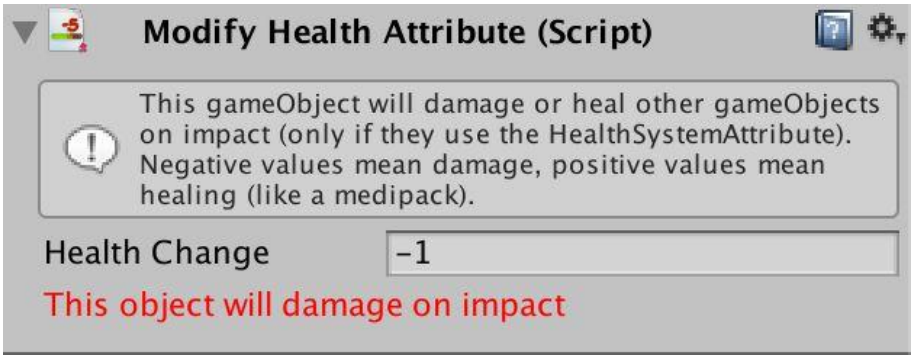


宇宙船と小惑星は緑色の境界線を示しています

PolygonCollider2Dが追加されたら、プレイボタンを押してみてください。宇宙船が小惑星を動かすことができるようになります。小惑星の重力を0に調整することを忘れないでください。また、小惑星のPolygonCollider2Dのパラメータである Friction(摩擦)、Angular Friction(回転摩擦)、およびMass

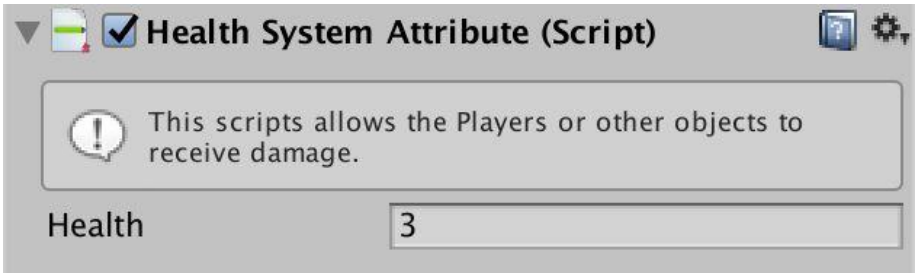
（質量）を自由に調整して、希望どおりの動きにすることもできます。Massを10に設定すると重くなり、宇宙船に触れても飛び散らないようになります。そう、大きい惑星は非常に重いに違いないのです！

そして、この小惑星を宇宙船の脅威となるようにしましょう。/ Scripts / Attributesにある**ModifyHealthAttribute**という名前のスクリプトを小惑星に追加します。



ModifyHealthAttributeのインスペクタ

それから、宇宙船をダメージが特定できるようにする必要があります。そのためには、宇宙船に**HealthSystemAttribute**という名前のスクリプト（/ Scripts / Attributesの下にあります）を加えます。



The HealthSystemAttributeのインスペクタ

HealthSystemAttributeは宇宙船にダメージ許容量（Health）を与えるだけでなく、その初期値を設定できます。最後に、小惑星を複製し（ショートカット：WindowsではCtrl + D、MacではCommand + D）、宇宙船の周りに小惑星群のフィールドを作成します。

Note

Prefab（プレハブ）

小惑星を複製する前に、小惑星をPrefab（プレハブ）にしたいと思うでしょう。そうすれば、後から小惑星を一度に簡単に編集することができるようになります。

Prefabについて知らないあるいは詳しく知りたい場合は、マニュアルもしくはビデオチュートリアルをご覧ください。

Prefabは、規模が大きなゲームを作ったりチームで共同開発したりする場合のUnityの基本的な概念ですが、今のところそれらを脇に置いておくこともできます。あなたはあなたの最初のゲームに集中し、そして時間があるときにそれらを改めて勉強するでも構いません。

ただし、宇宙船が攻撃を受けたときにフィードバックを受け取ることはありません。では、宇宙船のダメージ許容量を視覚化するためのUIを追加しましょう。/ Prefabsフォルダから**UserInterface** Prefabをシーンにドラッグ &ドロップします（うまくいかない場合は、ヒエラルキービューにドラッグ &ドロップします）。ゲームビューを見ると自動的にスコアとダメージ許容量のUIがポップアップします。

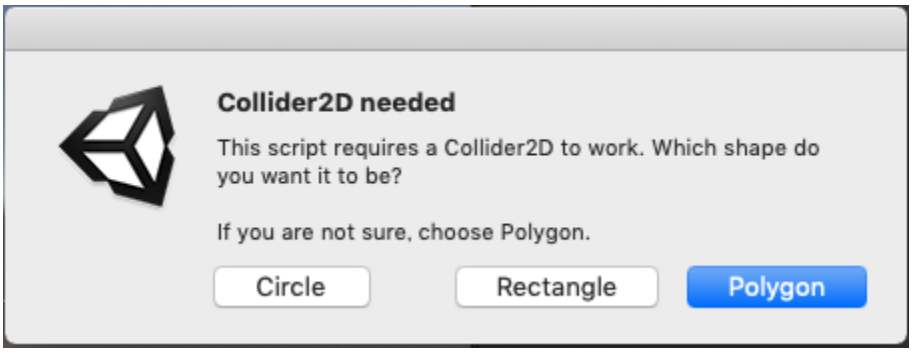
これでゲームをプレイすると、宇宙船が小惑星ぶぶつかるたびにダメージ許容量＝healthが1ポイント減ることがわかります！ ぶつかりすぎてhealthが0になるとゲームオーバーです！

今、小惑星が並ぶステージが出来上がり、小惑星にあまりぶつからず宇宙船を動かすことが実際にできるかをもう一度テストしてみましょう。ゲームにはある程度の難しさが必要ですが、難しすぎてもダメです。小惑星の位置を変更してゲームをテストすることで、あなたはゲームバランスに取り組んでおり、効果的にレベルデザインをしています。

ゴールを追加

さて、あなたが作ったゲームはどうですか？ それでは、小惑星に衝突することなく宇宙船が星を集めるようなゲームにしたい・・・としましょう。すべての星が集まったら、ゲームクリアになります。逆にあまりにも多く小惑星とクラッシュした場合はゲームオーバーです。

/ Images/GameElementsフォルダから星（Star）をシーンにドラッグ &ドラッグして追加しましょう。そして星に / Scripts / Attributeフォルダから**CollectableAttribute**スクリプトを追加します。その時、「Collider2D needed」のダイアログが表示されるので、Polygonを選択しましょう。**PolygonCollider2D**コンポーネントが自動で追加されます。



また、インスペクタでPolygonCollider2Dコンポーネントを見ると、Is Triggerにチェックが入っているはずです。もし入っていなければ、チェックを入れておきましょう。これで衝突を判定できるようになります。

CollectableAttributeスクリプトは星を収集可能なアイテムにして、星1つ獲得するとプレイヤーに1ポイントが与えられるようになります。

Note

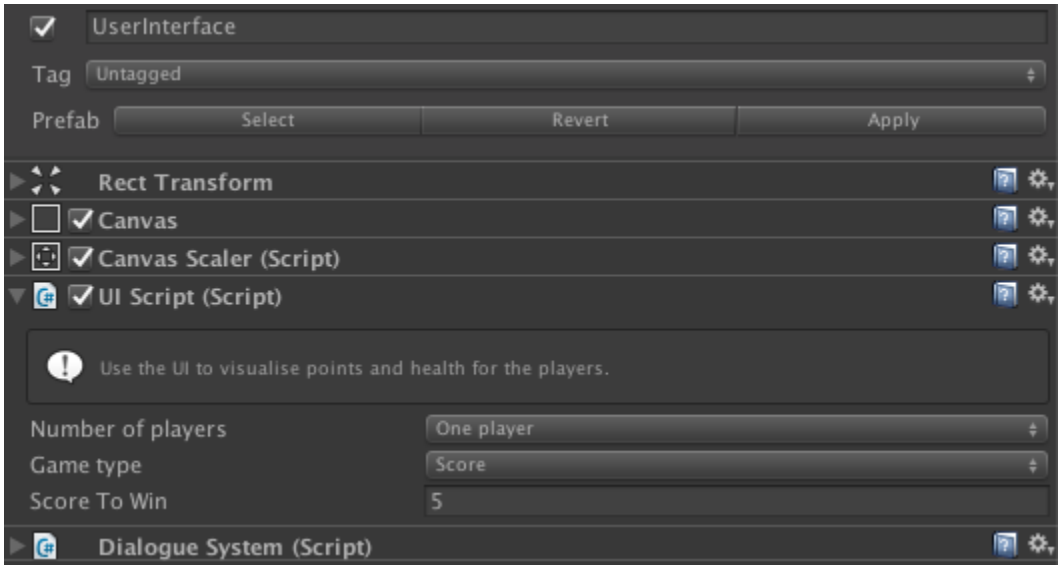
Triggers（トリガー）

トリガーは特別なタイプのコライダーです。トリガーはオブジェクトの形を無形化し、オブジェクト同士が触れることができないように思われます。しかしUnityは2つのオブジェクトが互いに接触したことを検出するので、結果としてアクションを実行することができます。

例えば、トリガーはプレイヤーがステージの最後に到達したかどうかを検出するのに非常に便利です。出口または終点の直前にトリガーを置き、プレイヤーがトリガーに触れると、You Winというメッセージを表示することができます。トリガーはプレイヤーには見えませんが、トリガーはあなた（開発者）に勝利状態を検出するためのロジックを提供します。

あなたが今ゲームをプレイするならば、あなたは星が宇宙船によって集められるのに気付くでしょう。この時点で星をプレハブにして（上記のプレハブについての注意を参照）、必要と思われる回数だけ複製することもできます。シーンに星を5つ置くとしましょう。そしてこれらを取りやすい位置や取りにくい位置に配置してみましょう。このようにして、ゲームに難易度をつけていきます。

最後に、ヒエラルキーからUserInterfaceを選択し、**UI Script**スクリプトのScore To Winの値が5であることを確認します。ゲーム内の星の数とScore To Winの値を一致させないと、正しくゲームが動きません。



もう一度プレイボタンを押してゲームをプレビューし、ゲームに勝利できるかを確認しましょう。5つ星をすべて獲得した場合は、「Player 1 wins!」というメッセージが表示されます。

おめでとうございます！　あなたは最初のゲームが作れました。さらに、宇宙船の制御具合や難易度調整、そしてレベルレイアウトを満足するまで調整してみましょう。

次のステップ

UnityとUnity Playgroundを初めて理解できたので、それを使ってより複雑なゲームを作ることができるはずです。一方で次のステップとして、Unity Playgroundをよりよく理解するためにUnity PlaygourndのGeneral Concepts（基礎概念）を見たいと思うでしょう。

これは覚えておいてください。Unity Playgroundは柔軟に設計しているので、あなた自身のゲームジャンルを考え出したり、既存のゲームをコピーしてみたりしましょう、何でもOKです！

Note

ゲームの作り方を学ぶとき、80年代からのレトロゲーム（たとえば、Asteroids（アステロイドズ）、Arkanoid（アルカノイド）、Space Invaders（スペースインベーダー）、Frogger（フロッガー）など）のコピー（真似）するのは、それらのゲームはシンプルなゲームデザインなので、非常に良い考えです。そしてあなたが上達するにつれて、あなたはより多くのゲーム要素を追加し、ゲームをより洗練することができます。

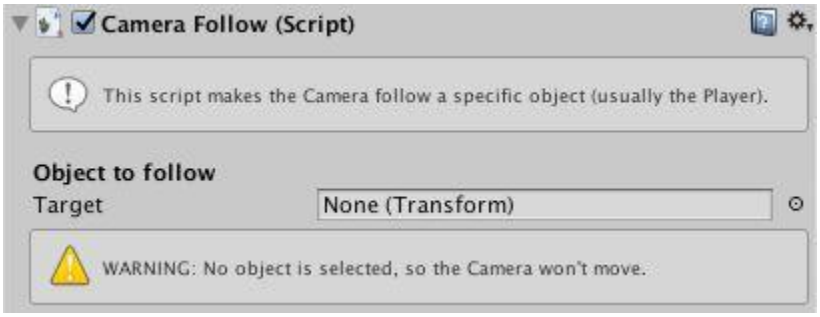
もしあなたがインスピレーションを必要とするならば、Examplesフォルダを開いて、その中のゲームを立ち上げてみましょう。ゲームオブジェクトを調べてそれらがどのように作られたのかを確かめてから、似たようなものを作ってみましょう。

General Concepts（基礎概念）

このセクションでは、Unity Playgroundのいくつかの基本的な概念を紹介します。まだ知らないことであれば、これから紹介する内容を使ってより複雑なゲームを作れるようになれます。このセクションを読む前に、先に紹介したGetting Started（入門）を行うことをお勧めします。

情報と警告

Unity Playgroundのすべてのカスタムスクリプトには、上部に小さな情報ボックスがあり、スクリプトが何をするのかを簡単に説明しています。



Camera Followスクリプトの情報と警告

同様に、多くのスクリプトにはいくつかの警告メッセージ（黄色い危険サイン付きのもの）があり、セットアップに問題があると表示されます。これらには注意してください。スクリプトが機能していない理由についての良いヒントを与えてくれるかもしれません。

衝突（Collisions）とトリガー（Triggers）

Unity Playgroundのほとんどのロジックスクリプトは衝突機能（Collisions）を使用します。これは、Collidersを持つ2つのオブジェクトが互いに接触したとき、またはColliderを持つオブジェクトが別のオブジェクト（Colliderも持っているがトリガーとしてマークされている）に侵入したときにイベントが発生することを意味します。

例えば、敵との衝突でダメージを与えたり、オブジェクトに触れたときにレベルを獲得したり、コインやパワーアップアイテムを集めたり、あるいはプレイヤーが正しい鍵を持っていたらプレイヤーがドアに触れたときに開いたりなど。

同様に、自分のキャラクタが別のキャラクタの前に入ったときに対話ラインをトリガーすることもできます。（Unity Learnのチュートリアルにあるroguelikeプロジェクトのように）

もしうまく動かなかったら、自分自身に問いかけてみましょう。「ちゃんとColliderを追加したか？」と。

タグ（Tags）

タグを使用すると、オブジェクトをカテゴリごとに分類できるので、スクリプトは正しいオブジェクトに触れた場合にのみアクションを実行することができます。タグなしではオブジェクトを区別することができません。

衝突と同様に、多くのスクリプトは正しく動作するためにオブジェクトにタグが正しく設定されていることが求められます。スクリプトが意図したとおりの動作しない場合は、リファレンスガイドを読んで、重要なタグがないかどうかを確認してください。

オリジナル画像のインポート

Unity Platgroundには、Imagesフォルダーの下にたくさんの素晴らしいスプライトがあります。しかし、あなたはここから2D画像を見つけるもよし、自分自身で作った2D画像ををを作る作るもよし、どんな2D画像も自由に使用できます！

Unity Playgroundで画像を使用するには、画像をAssetsフォルダのどこかにドラッグするだけです。画像はスプライトとしてインポートされ、使用する準備が整います。それをシーンまたはヒエラルキービューにドラッグ &ドロップするだけで新しいGameObjectが作成され、ゲームで使用できるようになります。

見栄えをよくするには、正方形ではない画像は透過にする必要があります。そうしないと背景が白く表示されます。画像を透過できる適切なファイル形式は.pngまたは.gifです。.jpgは透過情報を持てません。また、UnityはアニメーションGIFを再生できません。

Advanced Concepts（上級者向け概念）

あなたは先生ですか？　それとも教育者ですか？ このセクションはそんなあなたにとっての必読セクションです！ また、Unity Playgroundをよりよく理解したい上級ユーザーにもお勧めです。

チートシート

Unity Playgroundにどんなスクリプトがあるのか理解することは、一見したところでは難しいかもしれません。そこで、Unity Playgroundには各種チートシートが付属しています。これは、6ページに渡りすべてのスクリプトのアイコンと1行の説明があります。それらはカテゴリ（Movement、Gameplayなど）で編成され、色分けされています。

チートシートの内容

チートシートはUnityプロジェクト内の印刷用**.pdf**と画面用の個々の**.jpgs**の両方に含まれています。ユーザにUnity Playgroundが持っているすべての可能性を示すために、Unityに不慣れなユーザとのワークショップでは、このチートシートをいくつかプリントアウトしてそれらを配布するのはいいかもしれません。

これらに加えて、7ページ目にはタイプの異なるいくつかの追加課題があります。もしあなたがワークショップを運営しているなら、参加者に何か新しいことを試みたりあるいは制約を与えるためにこれらの追加課題を使用することもできます（ゲームジャムのテーマのように）。

プロジェクト構造

ワークショップでUnity Playgroundを使用するには、アセットストアから無料でダウンロードするか、インターネットアクセスが問題になる場合は、AssetsフォルダーとProjectSettingsフォルダーを手動で生徒に配布します。

アセットフォルダ

Documentationフォルダには、このドキュメントも含め、.pdf内のドキュメントが含まれています。このドキュメントの「Getting Started（入門）」を学習の出発点とし、さらに各スクリプトの詳細を調べたいときは「リファレンスガイド」を活用することができます。最後に、「チートシート」は印刷して学習者に配布することができ、スクリプトとその内容を一目で確認することができます。

Imagesフォルダには、キャラクター、敵、パーティクル、またはシーンの作成に使用できる画像アセットが含まれていますが、必要に応じて開発者は新しい画像を自由にインポートできます。

Unity Playgroundの核心となるスクリプトはScriptsフォルダにあります。それらのほとんどはシーンに出して動作させますが、動作させるために特定の方法でオブジェクトにタグを付けることを要求するものもあります（それらについての詳細はタグセクションでお読みください）。

Examplesというフォルダにはいくつかの小さなゲームがあります。学習教材として、またはカスタマイズの出発点として使用できます。

プロジェクトには_INTERNAL_という特別なフォルダがあります。その名前が示すように、Unity Playgroundの内部の仕組みを台無しにしたくない限り、それを触れられるべきではありません。プロジェクトの土台となるスクリプト、フォント、ギズモ、その他利用者が本当に気にするべきではないものが含まれています。しかし、Unity Playgroundが正しく機能するためには、このフォルダの中身はプロジェクトに含まれている必要があります。

このセクションは、その内部の仕組みをより深く理解するために、Unity Playgroundのいくつかの部分について少し詳しく説明します。学生を支援し、ゲームの問題を解決しなければならない教育者におすすめです。

タグ（Tags）

タグは、いくつかのスクリプトでオブジェクトをフィルタリングし、いつそれらの効果を生み出すかを決定するために使用されます。いくつかのスクリプトはオブジェクトにOnCollisionEnter2DまたはOnTriggerEnter2Dを自動追加しますが、他のもの（HealthSystemAttributeなど）は、オブジェクトがPlayerとしてタグ付けされているかどうかによって動作が異なります。

最初にProjectSettingsフォルダをインポートした場合は、いくつかの追加タグがすでに定義されています。具体的には・・・

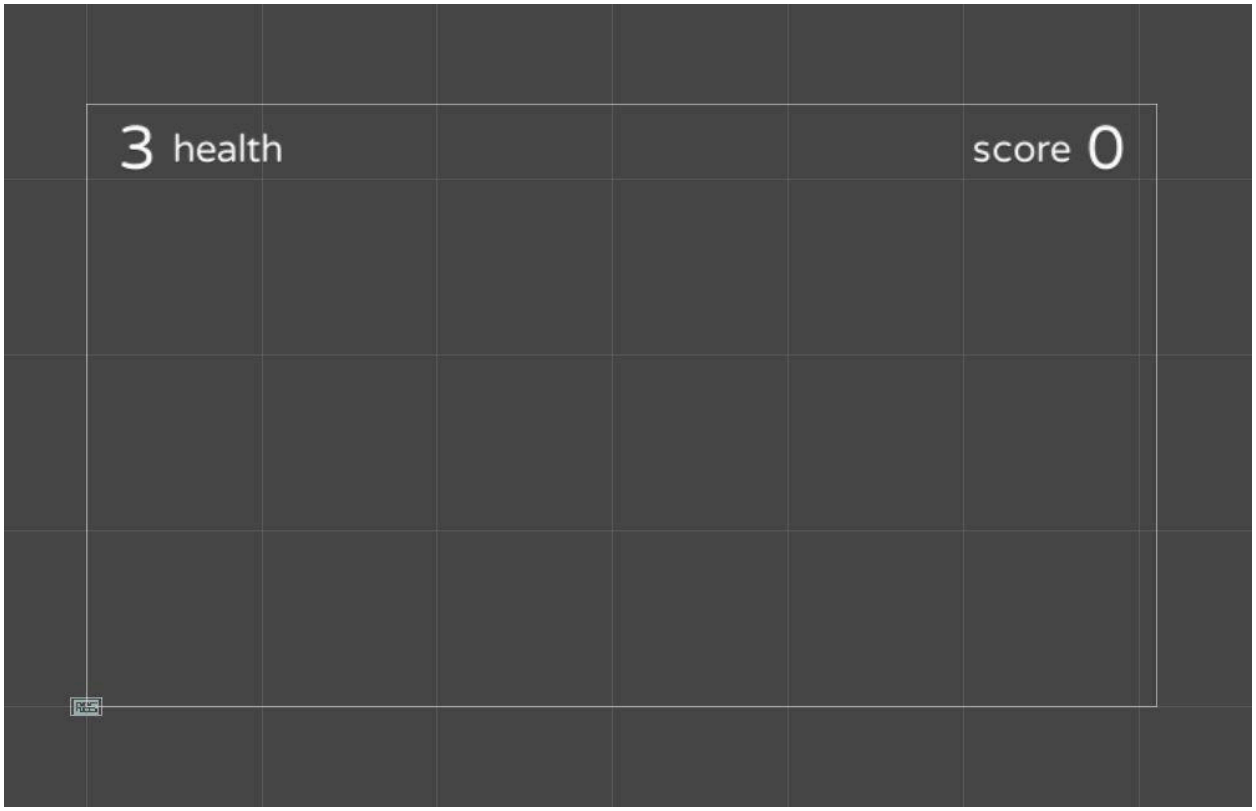
- プレイヤーごとのダメージやスコア・ポイントなど、2人のプレイヤーで使用するUIが適切に機能するようにするには、**Player**と**Player 2**のタグを使用する必要があります。
- **Enemy**は、現在どのスクリプトでも使用されていませんが、弾丸や武器の効果を生み出す敵を定義するのに役立ちます。
- **Bullet**は発射体で使います。
- **Ground**はプレイヤーがジャンプしたときにどれがグラウンド（地面）であるかをチェックするために使用されます。

Playerタグを除いて、多くのスクリプトでどのタグを探すかを定義できます。たとえば、Jumpスクリプトでは、グラウンド（地面）と見なされるものを定義するように求められます。したがって、必ずしもGroundタグを選択する必要はありません。

Conditionスクリプトでタグをフィルタリングできるので、何らかの方法でタグ付けされたオブジェクトと衝突したときだけ何かが起きる可能性があります。タグリストは自動的に更新されてタグのリスト全体を表示します。

ユーザーインターフェイス

ユーザーインターフェイス（UI）はPrefabsフォルダーに含まれているprefab（UserInterfaceというプレハブ）に実装されています。UIをシーンにドラッグ＆ドロップするだけで、プレイヤー1の状態と得点が自動的に表示されますが、UIScriptのNumber of playersでtwo playersを選択することもできます。



空のシーン背景に対するUIキャンバス

UserInterfaceのプレハブは、Game typeの項目でScore、Life、Endlessとゲームの種類を定義することもできます。モードに応じて、Game OverとYou Winの画面が表示されます。

- Scoreモードでは、プレイヤーが選択したスコアに達すると、勝利画面が表示されます。2人のプレーヤーが存在する場合、UIはスコアのみを表示し、スコアに到達した最初のプレーヤーが勝ちます。

- Lifeモードでは、プレイヤーの体力が0になるとゲームオーバーが表示されます。二人のプレイヤーで、両プレイヤーの体力が表示されます。勝利方法・条件はありません。
- Endlessモードでは、ゲーム終了画面は表示されず、ゲームを勝ち負けすることはできません。

Game typeのカスタマイズ

Conditionスクリプトを使用するときは、UIScriptのGameWonおよびGameOver関数をUnityEventに関連付けることができます。このようにして、衝突やその他のイベントを活用してカスタムの勝敗条件を作成できます。

同様に、AddOnePointおよびRemoveOnePoint関数を利用してスコアと同様のことを実行できます。

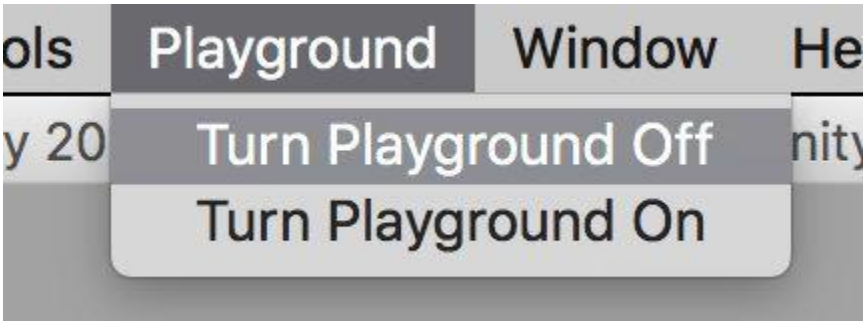
インスペクタのカスタマイズ

Unity Playgroundは、Playground自身のスクリプト（Move、Jumpなど）とデフォルトのUnityコンポーネント（Transform、Collider、SpriteRendererなど）の両方にカスタムインスペクタを多用しています。目標は、Unity UIから複雑さを取り除き、新しいユーザーがより簡単に作業できるようにすることです。

このため、いくつかの変数名も変更されました（たとえば、Rigidbody2DのDragはFrictionに変更されました）。

Unity Playgroundの無効化

上部のメニューバーからカスタムインスペクタをオンまたはオフにすることができます。これにより、Unity Playgroundが隠している項目を視覚化することができます。いつでも変更ができます。Unity Playgroundはインスペクタをカスタマイズしているだけなので、問題なく動作します。



トップメニューバーからUnity Playgroundをオフにする

このトリックは、Unity Playgroundが隠しているプロパティに一時的に変更を加える必要がある指導者・先生にお勧めです。（たとえば、任意のタイプのCollider形状の編集）

タイルマップ（Tilemaps）

画像アセットには、サンプルゲームで使用された一連のタイルスプライト（草、玉石、木）があり、UnityのTilemap機能を使用して独自のレベルを構築するために使用できます。

Tilemapについてもっと知りたい場合は、Unityのマニュアルに詳しいガイドがあります。LearnのWebサイトにも、それに関する優れたチュートリアルが豊富に用意されています。2D World Buildingの紹介ビデオから始めることをお勧めします。または、2D GamekitのPainting a Levelセクションで短い説明もあります。

Appendix

Contributing

For questions, suggestions, feel free to email Ciro Continisio (or get in touch through Twitter for quick comments). If you want to contribute to the project, check it out on Github , fork it, and create pull requests.

Note

All suggestions will be considered, but keep in mind that the goal of the Playground is **not** to have as many scripts and features as possible. Having a reasonable amount of content makes it more focused and easier to learn. We believe that once the learners have explored the entirety of the Playground and want more, it's time for them to move on and create their own game from scratch.

Credits

Unity Playground has been created by Ciro Continisio . Graphics by Stefano Guglielmana.

Special thanks to
Unity Technologies, and especially the Brighton Content team for helping with the final push to put the Playground on the Asset Store.
Kenney.nl for the free graphics which have been used for the first iteration of the Playground.
Dioselin Gonzalez , John Sietsma for believing in the project and using it in their workshops first.
Stine Kjærbøll for some precious early feedback, and the Note: for the logo design.
Nikoline Høgh and Nevin Eronde for more UX early feedback and enthusiasm.
Phil Jean for suggesting an important change which would be a turning point for the project.
Numerous contributors to the Github repository: Sophia Clarke , Ethan Bruins , Mark Suter , Jim"Jimbo" Picton , "Arche-san" , and all the others I'm forgetting right now!
The kids of Coderdojo Brighton and the attendees of New Employee Orientation at Unity, who have been the first involuntary test subjects.
Finally, all the teachers and educators who are using the Playground in any capacity today!