**بررسی مراحل توسعه‌ی بازی «بریک اوت!»**



**علی سجادی**

استاد مربوطه: **استاد گرامی**

دانشگاه آزاد اسلامی واحد شهرکرد

بهمن ماه 1397

**لینک بازی در کافه بازار:**

https://cafebazaar.ir/app/net.irannegah.breakout

**لینک سورس کد بازی در گیت هاب:**

https://github.com/asajadi84/Unity-Breakout

**فهرست مطالب:**

پیشگفتار ................................................................................................. 4

فلوچارت صحنه‌های بازی............................................................................ 5

assetهای استفاده شده در «بریک اوت!» ..................................................... 6

بررسی صحنه‌های بازی .............................................................................. 9

صحنه‌ی شروع بازی (ایندکس 0) ....................................................... 9

صحنه‌ی مرحله‌ی بازی (ایندکس 1) ................................................... 10

صحنه‌ی برنده شدن و باختن (ایندکس 2 و 3) .................................... 15

**پیشگفتار:**

بازی بریک اوت (Break Out)، یکی از نخستین بازی‌های ویدیویی شناخته شده در جهان است که برای اولین بار در دهه‌ی 70 میلادی و توسط کنسول آتاری به جهانیان ارائه شد. این بازی که با الهام از بازی Pong توسط استیو وازنیک و با ایده‌پردازی استیو جابز توسعه یافت توانست یکی از موفق‌ترین و پرفروش‌ترین عناوین بازی‌های آرکید شود. استیو وازنیک و استیو جابز بعدها کمپانی اپل را با همکاری یکدیگر تاسیس کردند.

بازی بریک اوت تا حدی شناخته شد که با وجود گذشتن سال‌های طولانی از توسعه‌ی آن، همچنان پرطرفدار و هیجان‌انگیز است. در طی این سال‌ها، توسعه‌دهندگان زیادی نسبت به ایجاد کپی‌های مختلفی از روی این بازی برای پلتفرم‌های مختلف اقدام نموده‌اند. در میان کپی‌های شناخته شده از روی بریک اوت، نمونه‌های زیادی وجود دارند که امکانات جدیدی به بازی اضافه کرده‌اند. امکاناتی نظیر هیولاهای مختلف، بسته‌های امتیازی که دریافت آن‌ها موجب اضافه شدن قابلیت‌های ویژه‌ای به پلیر شده و غیره؛ با این وجود بریک اوت حتی بدون امکاناتی که توسعه‌دهندگان جدید به آن اضافه کرده‌اند –در تولد چهل و چند سالگی‌اش– همچنان توجه گیمرهای نسل‌های مختلف را به خود جلب می‌کند. این بازی از زمان تولدش در میان پیکسل‌های درشت کنسول‌های دهه هفتادی آرکید تا به امروز در دیوایس‌های اندرویدی با گرافیک مدرن، راه خود را پیدا کرده است.

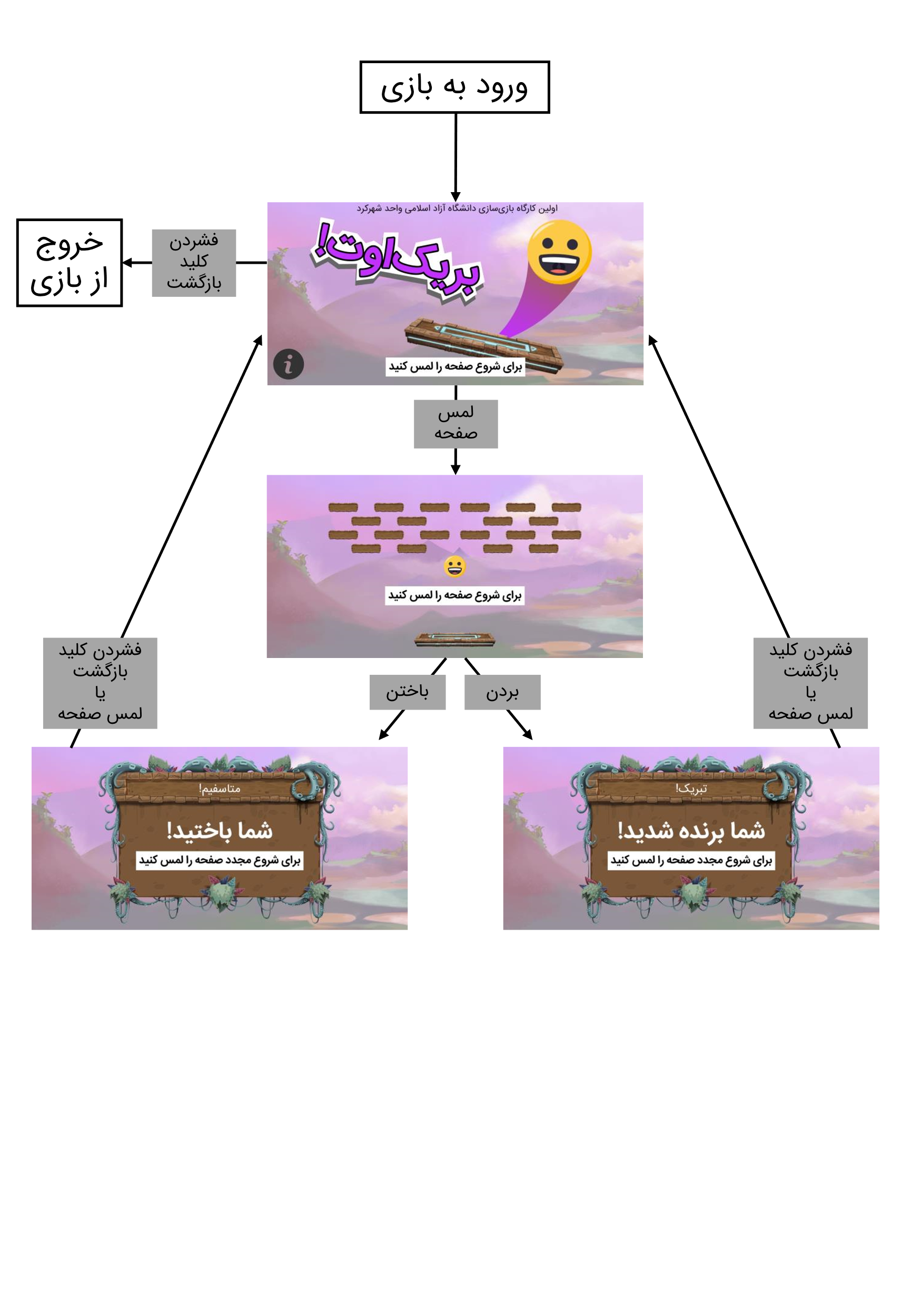
در این مقاله به بررسی خلاصه‌ی مراحل توسعه‌ی یک کپی از ایده‌ی این بازی می‌پردازیم. این کپی که با نام «بریک اوت!» در کافه بازار منتشر شده است در آزمایشگاه انجمن علمی کامپیوتر دانشگاه آزاد اسلامی واحد شهرکرد توسعه یافته است.­

برای توسعه‌ی این بازی از انجین یونیتی و زبان برنامه‌نویسی سی شارپ استفاده شده است. سی شارپ مورد استفاده در یونیتی در حال حاضر منطبق بر استانداردهای دات نت فریمورک 4.6 می‌باشد.

این مقاله بازی‌نامه (GDD: Game Design Document) بازی «بریک اوت!» نیست. چرا که بازی‌نامه پیش از توسعه‌ی بازی و به منظور راهنمایی توسعه‌دهندگان در حین روند ایجاد می‌گردد و نقشی مشابه فیلمنامه برای کارگردان دارد؛ در حالی که این مقاله پس از انتشار این بازی و به منظور توضیح روند توسعه‌ی «بریک اوت!» نوشته شده است.

**فلوچارت صحنه‌های بازی:**

بازی «بریک اوت!» از چهار صحنه تشکیل شده است که فلوچارت رویدادهایی که موجب سوییچ شدن بین این صحنه‌ها می‌گردند در زیر آورده شده است:

****

پیش از ایجاد هر صحنه، لازم است که assetهای موردنیاز آن صحنه از قبل ایجاد شده باشند که در بخش بعد به assetهایی که برای این صحنه‌ها ایجاد شد می‌پردازیم.

**assetهای استفاده شده در «بریک اوت!»:**

همان‌طور که می‌دانیم در مرحله‌ی پیش‌تولید هر بازی ویدیویی بایستی بخشی از assetهای آن آماده شود. asset به هر فایلی گفته می‌شود که در توسعه‌ی بازی مورد استفاده قرار می‌گیرد؛ از جمله تصاویر اشیای موجود در بازی (که به آن‌ها اسپرایت گفته شده و معمولاً با فرمت png خروجی گرفته می‌شوند)، فایل‌های صوتی شامل موسیقی و صداهای بازی، مدل‌های سه‌بعدی، فونت‌های مورد استفاده و...

در حالی که امکان ایجاد انواع خاصی از assetها در انجین بازی‌سازی وجود دارد (مثل انیمیشن، میکسر صوتی و متریال‌ها در یونیتی)، اما فرآیند ایجاد assetها مستقل از روند اصلی توسعه‌ی بازی‌سازی بوده و ممکن است حتی توسط افراد دیگری به جز تیم توسعه‌ی بازی صورت گرفته باشد.

برای مثال یک کانسپت آرتیست طراحی کاراکتر را بر عهده گرفته و این کار را در فتوشاپ یا procreate انجام می‌دهد و خروجی نهایی را با فرمت png در اختیار تیم توسعه قرار می‌دهند. این فرآیند به هیچ عنوان ارتباطی با بازی‌سازی نداشته و بنابراین از مراحل پیش‌تولید به شمار می‌رود؛ چرا که تیم توسعه‌دهنده تنها فایل آماده را وارد موتور بازی‌سازی می‌کند.

برای ایجاد assetهای تصویری «بریک اوت!» از برنامه‌ی ادوبی فتوشاپ استفاده شده است. همچنین assetهای صوتی نیز از وبسایت FreeSound (FreeSound.org) که صداهای مختلف برای بازی را به رایگان ارائه می‌کند برداشته شده‌اند. موسیقی بازی قطعه‌ای به نام Looking at the stars بوده که توسط Esther Garcia نواخته شده است.

در این قسمت به assetهای تصویری‌ای که برای ساخت این بازی طراحی شده است به تفکیک صحنه اشاره می‌کنیم. ضمن این که تعدادی از assetهای تصویری استفاده شده از نمونه‌های آماده‌ی موجود در asset store یونیتی برگرفته شده‌اند.

* **صحنه‌ی شروع بازی:** برای این صحنه سه asset تصویری طراحی شده است:
  + MainTitleBackground.png: به عنوان پس‌زمینه‌ی صحنه مورد استفاده قرار گرفته است.



* + MainTitleLogo.png: لوگوی بازی که به منظور اعمال انیمیشن چرخش بر روی آن به صورت مجزا از پس‌زمینه خروجی گرفته شده است.



* + MainTitleBottomText.png: به دلیلی پشتیبانی نکردن یونیتی از زبان فارسی، یکی از گزینه‌های پیش رو نوشتن متن موردنظر در یک فایل تصویری و وارد کردن آن به محیط یونیتی می‌باشد. این فایل شامل متن «برای شروع صفحه را لمس کنید» می‌باشد که در پایین صفحه قرار گرفته و انیمیشن بزرگ و کوچک شدن بر روی آن اعمال می‌شود.



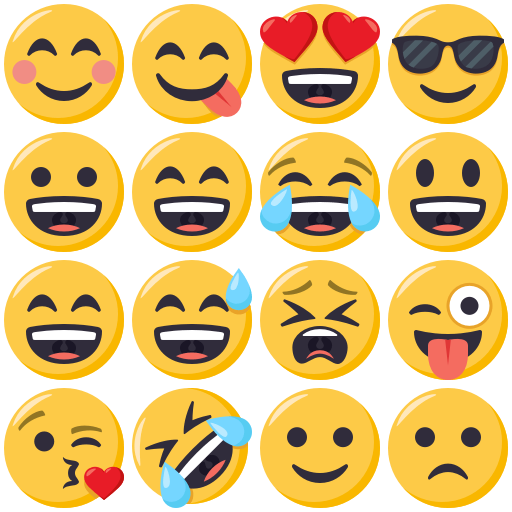
* **صحنه‌ی مرحله‌ی بازی:** برای این صحنه سه asset تصویری طراحی شده است:
  + SolidBackground.png: به عنوان پس‌زمینه‌ی صحنه مورد استفاده قرار گرفته است:



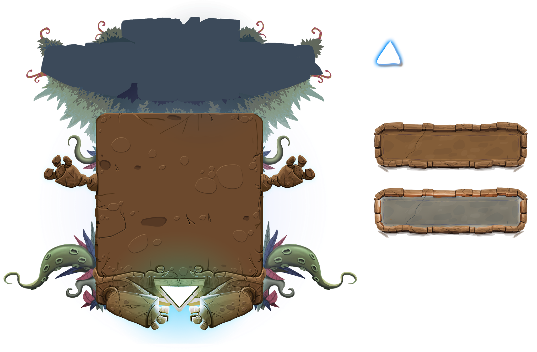
* + Bridge.png: اسپرایتی است که در مکان گیم آبجکت شناور (قابل کنترل توسط پلیر) رندر می‌شود.



* + EmojiOne.png: این فایل تصویری که شامل چندین صورتک است به صورت آماده از asset store یونیتی دریافت شده است. صورتک‌های خندان و ناراحت توسط ابزارهای داخلی یونیتی از داخل تصویر crop شده و برحسب نیاز به هرکدام در مکان گیم آبجکت توپ بازی رندر می‌شوند.



* + StartMenu.png: قسمت‌هایی از این اسپرایت توسط ابزارهای داخلی یونیتی crop شده و به عنوان تصویر آجر رندر می‌شود.



علاوه بر assetهای ذکر شده، در این صحنه از فایل MainTitleBottomText.png نیز مجدداً استفاده شده است.

* **صحنه‌های برنده شدن و باختن:** برای هر کدام از این صحنه‌ها یک asset تصویری ایجاد شده است که تلفیقی از پس‌زمینه، جعبه‌ی پیام و متن پیام به پلیر می‌باشند:

CongratsScene.png

GameOverScene.png

**بررسی صحنه‌های بازی:**

هر صحنه‌ی بازی در یونیتی، شماره‌ای تحت عنوان index داشته که برحسب ترتیب رندر شدن (محتمل) آن صحنه در زمان بازی کردن پلیر به آن اختصاص داده می‌شود. این شماره که در پنجره‌ی بیلد گرفتن از پروژه به فایل صحنه‌ها نسبت داده می‌شود در کدنویسی برای لود صحنه‌ی موردنظر مورد استفاده قرار می‌گیرد. index صحنه‌های «بریک اوت!» به ترتیب زیر است:

0: صحنه‌ی شروع بازی

1: صحنه‌ی مرحله‌ی بازی

2: صحنه‌ی برنده شدن

3: صحنه‌ی باختن

**صحنه‌ی شروع بازی (ایندکس 0):**



این صحنه اولین صحنه‌ای است که پلیر با اجرای بازی با آن روبرو می‌شود. در این صحنه اسپرایت پس‌زمینه به صورت استاتیک و دو اسپرایت دیگر، یعنی لوگوی بازی بازی و عبارت «برای شروع صفحه را لمس کنید» با انیمیشن‌هایی که در ادیتور یونیتی به آن‌ها نسبت داده شده است بر روی اسپرایت اصلی رندر می‌شوند.

این صحنه همچنین وظیفه‌ی پخش موسیقی بازی را نیز برعهده دارد. به این منظور به گیم آبجکت دوربین آن کامپوننت Audio Source شامل فایل صوتی موسیقی اضافه می‌شود.

طبق فلوچارت «بریک اوت!» پلیر هنگامی که در این صحنه قرار دارد باید با فشردن کلید Back اندروید (که Key Map آن با کلید Esc در ویندوز یکسان است) از محیط بازی خارج شده و با لمس صحنه (که معادل کلیک چپ در ویندوز است) به صحنه‌ی مرحله‌ی بازی (با ایندکس 1) وارد شود. به این منظور اسکریپتی با نام LoadSceneFromMainTitleToLevel ایجاد کرده و کد زیر را در آن می‌نویسیم:

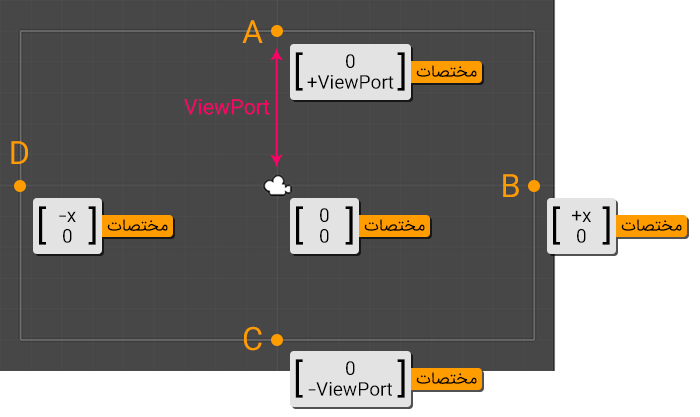


سپس اسکریپت فوق را به یک گیم آبجکت فعال در صحنه (برای مثال دوربین) نسبت داده تا در صحنه اعمال شود.

**صحنه‌ی مرحله‌ی بازی (ایندکس 1):**



در این صحنه قبل از هر عمل دیگری باید طول view port صفحه نمایش پلیر را محاسبه کنیم؛ **چرا که عرض view port مقداری ثابت است و در کامپوننت دوربین تعیین می‌شود، اما طول آن بسته به نسبت تصویر صفحه نمایش پلیر تغییر می‌کند.** در پروژه‌های دوبعدی یونیتی، Main Camera به صورت پیش‌فرض در حالت Orthographic قرار دارد. برخلاف حالت Perspective که در آن فضایی سه‌بعدی توسط دوربین رندر می‌شود، در حالت Orthographic صحنه به صورت یک صفحه در نظر گرفته می‌شود. در صورتی که گیم‌آبجکت دوربین در مبدا مختصات قرار داشته باشد (که در این حالت مرکزِ مستطیلِ خروجی رندر شده در مبدا مختصات قرار خواهد گرفت) مختصات سایر نقاط مهم در خروجی رندر شده توسط دوربین به صورت زیر خواهد بود (شدیداً توصیه می‌شود دوربین در شروع بازی در مبدا مختصات باشد):



در حالتی که دوربین به صورت Orthographic باشد ViewPort فاصله‌ی مرکز مستطیل خروجی رندر شده توسط دوربین که در پنجره‌ی Game مشاهده می‌شود (مبدا مختصات در تصویر) تا لبه‌ی آن را (نقطه A در تصویر) بر حسب واحد پیش‌فرض یونیتی که متر است تعیین می‌کند. این مستطیل در نمای صحنه با کادری خاکستری مشخص می‌شود (مثل تصویر بالا).

مهم نحوه‌ی نمایش خروجی رندر شده در نسبت‌های تصویر متفاوت: خروجی بازی بسته به رزولوشن و نسبت تصویر نمایشگرهای مختلف ممکن است برای هر پلیر از نظر اندازه و نمایش دادن/ندادن برخی گیم‌آبجکت‌های صحنه متفاوت باشد. فارغ از نسبت تصویر همواره نقطه‌ی A در وسط لبه‌ی بالایی نمایشگر و نقطه‌ی C در وسط لبه‌ی پایینی نمایشگر قرار خواهند گرفت. این بدان معنی است که خط فرضی AC همواره صفحه‌نمایش پلیر را به دو قسمت مساوی تقسیم می‌کند. بنابراین در اندازه‌های متفاوت نمایشگر اندازه‌ی خروجی رندر شده خود را طوری تطبیق می‌دهد که طول خط AC (به پیکسل) با عرض نمایشگر پلیر (به پیکسل) برابر شود.

در عرض نمایشگر پلیر **2 x ViewPort** واحد یونیتی رندر خواهد شد.

این مقداری ثابت است و تغییر نمی‌کند. اما برای تعداد واحدی که در طول نمایشگر پلیر رندر می‌شود عدد ثابتی وجود ندارد و بسته به نسبت تصویر نمایشگر تفاوت دارد. خروجی بازی پس از جای دادن عرض خود در عرض نمایشگر بیشترین مقداری را که می‌تواند در طول قرار دهد از دو طرف رندر می‌کند. یعنی از وسط تصویر آن‌قدر رندر را ادامه می‌دهد تا از چپ و راست به لبه‌های نمایشگر برخورد کند.

می‌دانیم که طول خط AC بر حسب واحد یونیتی برابر است با: **2 x ViewPort**

پس اگر عرض نمایشگر پلیر به پیکسل را ScreenY در نظر بگیریم هر واحد مربع یونیتی بسته به رزولوشن نمایشگر کاربر با ابعاد [ScreenY / (2 \* ViewPort)] x [ScreenY / (2 \* ViewPort)] پیکسل رندر می‌شود.

حال اگر طول نمایشگر پلیر به پیکسل را ScreenX در نظر بگیریم بنابراین در طول نمایشگر پلیر ScreenX / [ScreenY / (2 \* ViewPort)] واحد یونیتی رندر می‌شود. اگر این عدد را به ۲ تقسیم کنیم می‌توان x+ و x- تصویر بالا را محاسبه نمود.

برای محاسبه‌ی این مقدار از اسکریپتی به نام ViewPortCalculator استفاده می‌کنیم. از دیگر وظایفی که به این اسکریپت داده شده است مکاندهی چهار دیوار نامرئی در چهار حاشیه‌ی صفحه‌نمایش پلیر بوده تا توپ از صفحه خارج نشود:



پس از انجام این کار می‌خواهیم گیم آبجکت شناور به دنبال ماوس حرکت کند؛ اما با رعایت دو شرط: اول آن که تنها طول آن با طور فلش ماوس (یا انگشتی که تاچ می‌کند) هماهنگ باشد و عرض آن ثابت بماند و دوم آن که امکان خارج شدن از صحنه را نداشته باشد. برای پیاده سازی این قابلیت‌ها اسکریپتی به نام PaddleMovement ایجاد کرده و آن را به گیم آبجکت شناور نسبت می‌دهیم:



برای این که هنگام برخورد توپ به شناور، به آن نیروی مشخصی وارد شود تا ساکن نشود نیز اسکریپتی به نام PaddleAddForce ایجاد کرده و پس از نوشتن کدهای زیر در آن، اسکریپت را به شناور نسبت می‌دهیم:



البته این اسکریپت باید در Inspector مقداردهی ثانویه شود.

در این صحنه از تعدادی آجر استفاده شده است که همگی یک prefab بوده و با اولین برخورد پلیر صدای ضربه پخش شده و شکل آن‌ها تغییر می‌کند و با دومین برخورد نابود شده و صدای خرد شدن پخش می‌شود. برای پیاده کردن این قابلیت اسکریپتی به نام BrickHandler ایجاد کرده و به prefab آجر نسبت می‌دهیم:



در این بازی، بردن هنگامی اتفاق می‌افتد که هیچ آجری در صحنه باقی نمانده باشد (تعداد آجرهای باقی مانده صفر باشد) و باخت هنگامی اتفاق می‌افتد که توپ به حاشیه‌ی پایین صفحه (که در آن‌جا یک گیم آبجکت نامرئی تنها برای تشخیص برخورد قرار داده‌ایم) برخورد کند. بنابراین به یک متغیر برای نگهداری تعداد آجرهای باقی مانده در صحنه احتیاج داریم. به این منظور اسکریپتی به نام NumberOfBricksLeftController استفاده می‌کنیم. این اسکریپت احتیاجی ندارد که به یک گیم آبجکت متصل شده و بنابراین لازم نیست از کلاس MonoBehaviour ارث‌بری کند:



دو اسکریپت بالا تعداد آجرهای باقی مانده را نگهداری کرده و هر بار که یک آجر نابود شد یکی از این عدد کم می‌کنند تا هنگامی که این عدد به صفر رسید صحنه‌ی برنده شدن پلیر (با ایندکس 2) لود شود.

برای تشخیص باخت پلیر نیز از اسکریپتی به نام BallFell استفاده می‌کنیم که در صورت برخورد به حاشیه‌ی پایینی صحنه‌ی باخت (با ایندکس 3) را لود می‌کند. این اسکریپت به گیم آبجکت نامرئی پایین صفحه نسبت داده می‌شود:



**صحنه‌ی برنده شدن و باختن (ایندکس 2 و 3):**





در این دو صحنه (که پس از برد یا باخت پلیر لود می‌شوند) قصد داریم پلیر با لمس صفحه به صحنه‌ی شروع بازی باز گردد. بنابراین اسکریپتی به نام LoadScenesFromMessageScenesToMainTitle استفاده می‌کنیم:



پایان