GODOT

```gdscript
extends Node

@rpc
func my_func():
    print("Function called by the multiplayer authority")

@rpc(any_peer)
func my_func_any_peer():
    print("Function called by a remote peer")

@rpc(call_local)
func my_func_local():
    if multiplayer.get_remote_sender_id() == multiplayer.get_unique_id():
        print("Function called locally")
    else:
        print("Function called by the multiplayer authority")

@rpc(any_peer, call_local, unreliable_ordered, 1)
func my_func_ordered():
    print("My ordered RPC on the first channel")
```

کانال ،RPC نحو :Godot 4.0 چند نفره در ها، سفارش

توسط: فابیو الساندرلی

سپتامبر 25 2021

گزارش پیشرفت

این مقاله مربوط به **سپتامبر 2021** است ، ممکن است برخی از مطالب آن قدیمی باشد و دیگر دقیق نباشد.
. می توانید اطلاعات به روز موتور را در اسناد رسمی بیابید

سلام گودوترز! زمان برای به روز رسانی دیگری در مورد شبکه چند نفره Godot 4.0 است.

اخیراً ما واقعاً مشغول کار بر روی پایه کلاس‌های شبکه و چند نفره بوده‌ایم، و چند ویژگی جدید برای شروع RPC صحبت در مورد آنها وجود دارد. در این پست، با نمایش برخی از نحو و ویژگی های جدید می کنیم.

سایر مقالات این مجموعه شبکه گودو 4.0 را ببینید:

1. و به‌روزرسانی‌های حالت RSET، روی سرورها :Godot 4.0 چند نفره در
2. کانال ها، سفارش ،RPC نحو :Godot 4.0 چند نفره در (شما اینجا هستید)
3. Godot 4.0: ENet wrappers، WebRTC چند نفره در
4. Godot 4.0: Scene Replication (قسمت 1) چند نفره در

پیکربندی RPC ساده شده

و گیج کننده `master` اول از همه، بسیاری از کاربران متوجه شدند که کلمات کلیدی قدیمی `puppet` و `master` هستند.

کلیدی به این معنی است که یک تابع را می توان در "مستر شبکه" فراخوانی کرد، در `master` کلمه که یک تابع را می توان فقط روی همتاهای "غیر اصلی" فراخوانی کرد. علاوه بر این، `puppet` حالی می‌توانست در جای خود `remote` کاربرد بسیار کمی داشت، زیرا `master` کلمه کلیدی قدیمی بدون هیچ تلاشی استفاده شود.

با آموختن از این موضوع، تصمیم گرفتیم یک حاشیه نویسی یکپارچه با چند پارامتر اختیاری `@rpc` داشته باشیم .

قدرت

```
@rpc
func my_rpc():
```

```
        print("RPC called.")
```

تماس‌ها فقط از **طرف مرجع چند نفره** ، که به‌طور پیش‌فرض سرور است، `@rpc` ،به‌طور پیش‌فرض مجاز است. شما می توانید اختیاری چند نفره را بر اساس هر گره از طریق روش تنظیم کنید `Node.set_multiplayer_authority()`.

کلمه کلیدی قدیمی رفتار می کند `puppet` حاشیه نویسی به تنهایی مانند `@rpc` ،از این نظر.

```
@rpc(any_peer)
func my_rpc():
        print("RPC called by: ", multiplayer.get_remote_sender_id())
```

حاشیه نویسی همچنین می تواند پارامترهای اختیاری داشته `@rpc` ،همانطور که در بالا ذکر شد توسط هر همتای متصل قابل فراخوانی RPC ، `any_peer` باشد. اگر یکی از آن پارامترها باشد کلمه کلیدی قدیمی رفتار می کند. شما می توانید شناسه همتا را که در `remote` خواهد بود و مانند روش `MultiplayerAPI.get_remote_sender_id()` حال برقراری تماس است از طریق دریافت کنید.

```
@rpc(any_peer)
func my_rpc():
        var peer_id = multiplayer.get_remote_sender_id()
        if peer_id == get_multiplayer_authority():
                # The authority is not allowed to call this function.
                return
        print("RPC called by: ", peer_id)
```

در `master` هیچ جایگزین مستقیمی برای کلمه کلیدی که به ندرت استفاده می شود وجود ندارد می توان با اضافه کردن یک چک اضافی در برابر شناسه فراخوانی `@rpc(any_peer)` ،این موارد شده، همانطور که در بالا انجام شد، استفاده کرد.

فراخوانی توابع به صورت محلی

```
@rpc(call_local)
func my_sync_rpc():
        print("RPC called")
```

یک تابع خاص نیز به صورت محلی ،RPC در گودو، می توان به موتور دستور داد که هنگام ارسال فراخوانی شود.

این کار با استفاده از کلیدواژه‌های اختصاصی‌تر (مانند، و غیره) انجام ، x. در گودو 3 اکنون یک پارامتر اختیاری از sync، Godot 4.0 در . remotesync puppetsync می‌شود @rpc حاشیه نویسی ها است.

دارند که هر همتا می RPC پارامترها نیازی به ترتیب خاصی ندارند، بنابراین معنایی معادل تعریف یک @rpc(call_local, any_peer) . @rpc(any_peer, call_local) تواند آن را فراخوانی کند ها نیز به صورت محلی روی همتای ارسال کننده اجرا،RPC پارامتر sync به لطف می شوند.

ساده شده RPC تماس های

ها داشتیم: قابل اعتماد و غیرقابل اعتمادRPC ما قبلاً 2 حالت انتقال مختلف برای ، x. در گودو 3.

آن را به طور قابل اعتماد منتقل می کند، در حالی که تماس ،rpc("my_func") تماس، rpc_unreliable("my_func") آن را به طور غیر قابل اعتماد منتقل می کند.

با این حال، در بیشتر موارد، همیشه می خواهید از همان حالت انتقال استفاده شود (به استثنای معدود).

```
@rpc(unreliable)
func my_unreliable_rpc():
        print("RPC called.")
```

نویسی تبدیل @rpc در گودو 4.0 تصمیم گرفتیم حالت انتقال را نیز به عنوان یک پارامتر حاشیه کنیم.

تابع اختصاصی (هنوز پیاده‌سازی نشده) پیکربندی را rpc_raw همچنان می‌توانید با استفاده از یک خاص لغو کنید RPC برای یک.

کانال و سفارش

چند نفره در گودو، 4.0 معرفی کانال ها و حالت انتقال سفارشی است API دو ویژگی جدید.

کانال ها

. از مفهوم *کانال* پشتیبانی می‌کنند ،WebRTC و ENet بیشتر پروتکل‌های شبکه بلادرنگ، از جمله

در صورت تمایل می‌توانید کانال‌هایی مانند جریان‌های مجزا در یک اتصال مشابه یا حتی اتصالات جداگانه به یک همتای راه دور را در نظر بگیرید. هر کانال به طور مستقل از یکدیگر عمل می کند، و مانند رودخانه هایی که با سرعت های مختلف جریان دارند، پیام های قابل اعتمادی که در کانال های مختلف ارسال می شوند، ممکن است به ترتیب متفاوتی برسند.

این ممکن است در ابتدا یک محدودیت به نظر برسد، اما در واقع قدرت واقعی آنهاست.

را به روشی قابل اعتماد ارسال می کنید، پروتکل باید آن را پیگیری کند و (RPC) هر بار که پیامی منتظر بماند تا مشتری قبل از ارسال پیام های بیشتر، دریافت آن را تأیید کند. در حالی که پروتکل‌ها از تکنیک‌های زیادی برای بهینه‌سازی این فرآیند استفاده می‌کنند (مثلاً بافر کردن چند پیام)، این امر به‌طور اجتناب‌ناپذیر تأخیر را معرفی می‌کند.

خواهید داشت که کاملاً با بقیه نامرتبط هستند (مثلاً چت بازیکن). RPC در بازی خود، احتمالاً تعدادی ها کاملاً با بقیه بازی هماهنگ باشند (در حالی که نظم داخلی را حفظ می کنند). RPC لازم نیست این در این موارد، به خصوص هنگام انتقال حجم بیشتر داده، استفاده از یک کانال مجزا یک راه کارآمد برای کاهش تأخیر و کاهش خطر قطع اتصال است.

```
@rpc(any_peer, 1)
func my_chat_func():
        print("RPC received on channel 1.")
```

این بهینه‌سازی‌ها را آسان‌تر می‌کند و به شما امکان می‌دهد با ارسال یک عدد صحیح Godot 4.0 به‌عنوان آخرین پارامتر حاشیه‌نویسی، کانال دیگری را برای استفاده غیر از پیش‌فرض مشخص کنید `@rpc`.

این همچنین با ویژگی جدید دیگر، حالت انتقال "سفارش" مفید است.

مرتب سازی

```
@rpc(unreliable_ordered)
func my_ordered_rpc():
        print("Ordered RPC received")
```

های غیرقابل اطمینان تضمین نمی شوند که به ترتیب وارد شوند. اگر سرور ابتدا RPC ،به طور کلی A ابتدا دریافت کند و سپس B یک کلاینت *می تواند* ، B و سپس پیام را ارسال کند A پیام.

غیرقابل اعتماد است که همچنان پیام‌های دریافتی را به ترتیب RPC سفارش‌شده» یک» RPC به‌طور خودکار هر پیامی را که سرور ، B صحیح تضمین می‌کند. یعنی، اگر کلاینت‌ها را دریافت کنند دور می‌اندازد (اگر در زمان بعدی دریافت شده باشد A از جمله) قبل از آن ارسال کرده است.

نکته احتیاط: حالت انتقال سفارشی ابزار قدرتمندی برای کاهش عملکرد بیشتر اتصالات شبکه است، اما اگر به درستی استفاده نشود، جنبه منفی آن افزایش بالقوه تلفات بسته است.

```
@rpc(unreliable_ordered, 1)
func _update_players_state(state):
        # Code to update the state of the players
        pass

@rpc(unreliable_ordered, 2)
func _update_enemies_state(state):
        # Code to update the state of the enemies.
        pass
```

هنگام استفاده از حالت انتقال سفارشی، به شما توصیه می شود که پیام های ناهمگون را از طریق یک کانال **ارسال نکنید** .

در قطعه کد بالا، ما در حال طراحی یک بازی هستیم که در آن مشکلی نداریم که حالت های بازیکن و دشمن در کلاینت کمی تغییر کند. با این حال، ما می خواهیم که هر ایالت فقط در صورتی به روز شود که حالت دریافتی جدیدتر باشد (از این رو حالت "سفارش داده شده").

استفاده کنیم زیرا 2 سفارش جداگانه می RPC در این صورت باید از کانال های مختلف برای 2 خواهیم. در غیر این صورت، به‌روزرسانی «بازیکنان» ممکن است حذف شود، زیرا وضعیت «دشمن» جدیدتری قبلاً دریافت شده است، که آن چیزی نیست که ما می‌خواهیم.

کار آینده

چیزهای جدید و هیجان انگیز زیادی وجود دارد که می توان در مورد آنها صحبت کرد، از بازسازی گرفته تا کارهای سطح بالای انجام ENet سنگین کلاس های شبکه و افشای بیشتر توابع سطح پایین شده روی تکثیر صحنه جدید. در ادامه با ما همراه باشید :)

کار مرجع

- بازساز RPC
- حاشیه نویسی روابط عمومی `@rpc`
- حذف استاد/عروسک، اقتدار (تنها اخیراً ادغام شده است زیرا به توافقی در مورد نام‌گذاری‌های جدید نیاز داشت).

حکومت

تیم ها

منابع اضافی

کتابخانه دارایی

مستندات

مخزن کد

با ما تماس بگیرید