

# PROTON BUS SIMULATOR

[www.protonbus.com.br](http://www.protonbus.com.br)

# Mods de Mapas: Fase 3

Atualizado em setembro de 2021

por Marcos Elias

<b>Introdução</b>	<b>3</b>
<b>Grupos no Facebook</b>	<b>4</b>
Grupos para usuários:	4
Dica de um bom site de onde obter texturas para os mods:	4
<b>Observações referentes às fase 1 e 2</b>	<b>5</b>
<b>A estrutura de arquivos do mod de mapa</b>	<b>7</b>
<b>Configurando os pontos de entrada</b>	<b>13</b>
Definição da característica da rota: rodoviária ou urbana?	19
É letreiro reservado, garagem ou algum que proíba a entrada?	20
<b>Configurando os passageiros</b>	<b>21</b>
A configuração dos pontos do modelo 3D	22
A configuração dos pontos no txt	30
<b>Configurando os pedestres</b>	<b>34</b>
Agora vamos à prática!	38
<b>Configurando os veículos do tráfego</b>	<b>43</b>
<b>Configurando as pinturas dos ônibus</b>	<b>46</b>
<b>Configurando os trens</b>	<b>47</b>
<b>Fase 3: Semáforos</b>	<b>50</b>
Configurações gerais do motor do semáforo	53
Configurações de cada tick	55

Configurações de luzes reais	56
Colocando os semáforos no mapa no 3D	57
Configurando o trigger do bloqueio para os veículos	59
<b>Fase 3: Luzes dos postes e ambientes</b>	<b>62</b>
Configurações das luzes reais	64
Configurações das luzes fakes	65
Observação sobre as luzes	66
<b>Fase 3: GPS</b>	<b>69</b>
<b>Comandos e nomes especiais nos modelos 3D</b>	<b>71</b>
Peças transparentes, para grades, árvores	71
Peças com colisor, para ruas, calçadas, plataformas, pisos	71
Peças invisíveis com colisor, para bloqueios, pisos complementares	72
Peças sempre acesas, como painéis luminosos de propagandas	73
Indicador de área de terminal ou ponto final, para não reclamarem da velocidade baixa	73
Indicador de ponto final, para todo mundo desembarcar	74
Fase 3: objetos aleatórios que podem estar ou não no lugar	75
Detalhes extras, para deixar o mapa mais leve	77
Configuração das texturas cintilantes no chão e muros, telhados	78
<b>Extras - Dicas de otimização</b>	<b>79</b>
Quais os impactos se fizer um mapa muito grande?	82
A complexa questão de mapas abertos na Unity	84
<b>Ferramentas de Debug</b>	<b>85</b>
<b>Nunca use imagens com dimensões acima de 2048 pixels de largura ou altura para os celulares atuais!</b>	<b>86</b>
<b>Sempre teste no celular, se possível!</b>	<b>88</b>
<b>Isso é tudo por agora!</b>	<b>89</b>

# Introdução

A **fase 3** do sistema de mods de mapas do **Proton Bus** permite ter passageiros, tráfego, semáforos e outras coisas a mais no jogo! Isso eleva ele a um outro patamar, literalmente dá para separar o Proton antes e depois deste recurso. É quase que como um novo jogo!

**IMPORTANTE! PARA PODER USAR OS SEMÁFOROS E OUTROS RECURSOS DA FASE 3, ALTERE O PARÂMETRO `mapModVersion=3` NA SEÇÃO `[map]` DO INI. SEM ISSO OS NOVOS COMANDOS SERÃO IGNORADOS PARA NÃO AFETAR A PERFORMANCE DOS MAPAS DAS FASES 2 OU 1.**

Devido às questões de tempo, não é possível dar suporte individual no processo de conversão e edição. Recomendamos tentar por você mesmo, vendo como foram feitos os outros mapas e explorando as estruturas dos arquivos. Prefira postar dúvidas e problemas relacionados nos grupos do jogo no Facebook, assim as soluções atendem a todos, e também outros produtores de mods podem lhe ajudar.

Caso ainda não esteja inscrito, entre no canal do produtor: [www.youtube.com/marquinhosxp](https://www.youtube.com/marquinhosxp)

Alguns vídeos e tutoriais são postados lá.

Página oficial do PBSU no Facebook: <https://www.facebook.com/protonbusoficial>

E do PBSR: <https://www.facebook.com/protonbusrod.oficial/>

# Grupos no Facebook

Grupo exclusivo para criadores de mods:

<https://www.facebook.com/groups/pbsmods/>

Por favor, evite entrar neste caso seja apenas jogador, para não tumultuar com postagens fora do tema (que serão excluídas)!

## Grupos para usuários:

Nestes sim, postagens livres de prints, dicas, vídeos, tutoriais diversos, etc, do ponto de vista dos jogadores:

Grupo internacional:

<https://www.facebook.com/groups/1264008273712815/>

Grupo brasileiro oficial, da equipe:

<https://www.facebook.com/groups/2238192089740569/>

Grupos de fãs mais antigos:

<https://www.facebook.com/groups/624645387735090/>

<https://www.facebook.com/groups/1678009835826675/>

Dica de um bom site de onde obter texturas para os mods:

<http://www.textures.com>

## Observações referentes às fase 1 e 2

Antes de iniciar um mapa nas fases mais avançadas, é necessário ter dominado a produção do nível básico para a fase 1. Se você ainda não tem familiaridade, não se preocupe, é possível seguir com este guia acompanhando o mapa de exemplo. O maior requisito é ter conhecimentos básicos de Blender, pois usaremos ele para exportar os mapas.

A documentação antiga referente à fase 1 está aqui, para mapas:

<http://blog.protonbus.com.br/2019/09/experimental-mods-de-mapas-no-pbs.html>

E para ônibus:

<http://blog.protonbus.com.br/2018/09/primeira-fase-do-sistema-de-mods-de.html>

É sempre bom estar com a versão mais recente do jogo! As antigas não são mais suportadas, o beta está em constante evolução, então sempre consideramos a mais recente. Você encontra as compilações mais recentes em <http://pbsu.busmods.com> para o urbano e <http://pbsr.busmods.com> para o rodoviário.

Basicamente usamos o **Blender 2.79** para produzir o cenário, exportamos o modelo no formato 3ds e criamos a estrutura de pastas e arquivos para o sistema do Proton Bus.

### IMPORTANTE: USE O BLENDER 2.79!

O exportador necessário para o formato atual do jogo não funciona com o Blender 2.8, aparentemente ele foi descontinuado e ninguém até hoje atualizou. Quem sabe futuramente ele volte a funcionar, tem um pessoal tentando. Você pode baixar o Blender 2.79 aqui:

<https://download.blender.org/release/Blender2.79/>

A versão mais recente normalmente é a que leva um “b” no nome. A plataforma é indiferente, vai depender do seu sistema (Windows, Linux ou Mac) e arquitetura (32 ou 64 bit).

Se você já iniciou o mapa no 2.8, a recomendação seria exportar para algum formato intermediário que o 2.79 possa abrir e então exportar a partir dele para o jogo.

### IMPORTANTE: MODIFIQUE O EXPORTADOR DE 3DS PARA O PROTON!

O exportador padrão de 3ds do Blender trunca os nomes dos objetos e das texturas em 12 caracteres. Normalmente precisaremos de mais do que isso para evitar erros nos nomes de algumas peças e/ou nas texturas.

Download do plugin do Blender alterado para exportar (arquivo para o Blender 2.79):

[http://proton.viamep.com/coisas/export\\_3ds\\_protonbus\\_blender279.zip](http://proton.viamep.com/coisas/export_3ds_protonbus_blender279.zip)

TUTORIAL DE COMO ALTERAR O PLUGIN: <https://youtu.be/0EokFmSjGdA>

Se preferir, em vez de baixar este, altere o arquivo **export\_3ds.py** da pasta **scripts\addons\io\_scene\_3ds** do seu Blender (localize a pasta dele lá na arquivos de programas, ou onde extraiu, caso tenha usado o zip sem instalador).

Procure por **[:12]** e troque o **12** por um número bem maior, como **999**. Ele é o limite de caracteres. Sem isso a exportação para o jogo pode falhar, pois ele limita a 12 caracteres por padrão os nomes dos objetos e texturas. Essa etapa é muito importante, esteja avisado!

**DICA: AO BAIXAR O BLENDER VOCÊ PODE OPTAR PELAS VERSÕES “PORTABLE” EM ZIP, PARA MANTER VÁRIAS VERSÕES AO MESMO TEMPO. GERALMENTE O INSTALADOR SUBSTITUI O BLENDER E FICA UM SÓ, MAS USANDO OS ZIP, DÁ PARA TER VÁRIOS.**

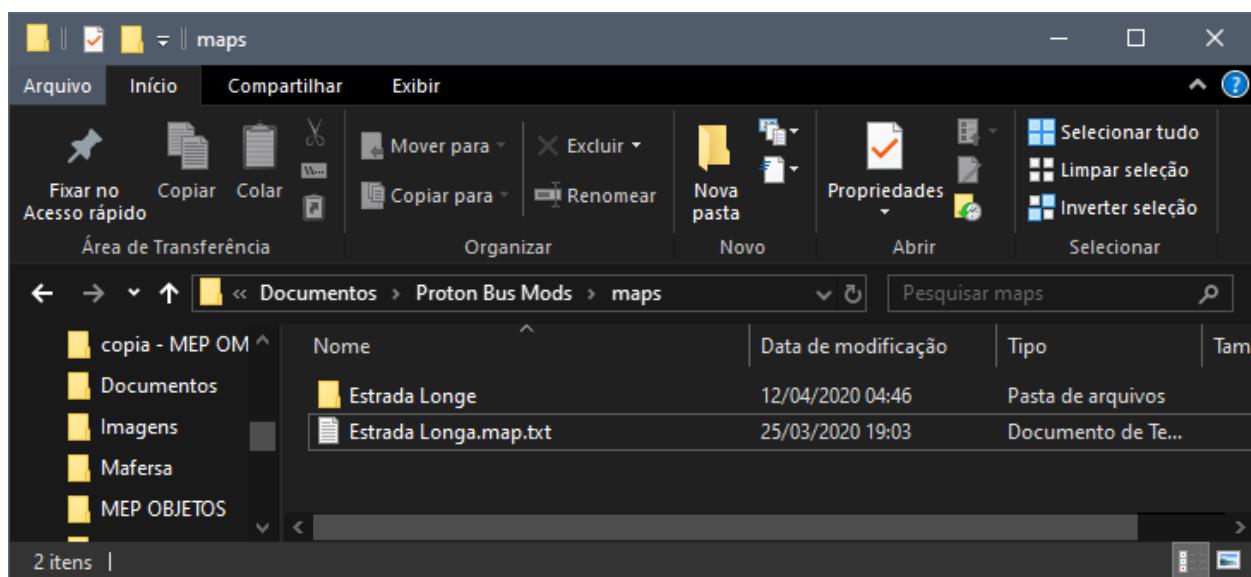
A fase 3 adicionou semáforos, luzes e alguns outros detalhes. Eles foram adicionados ao final deste documento, antes das recomendações de performance e limitações conhecidas.

**UM PEDIDO: AO PUBLICAR SEU MAPA PARA DOWNLOAD, POR FAVOR, USE O FORMATO ZIP SEM UMA SENHA! ASSIM O INSTALADOR DO JOGO CONSEGUIRÁ ABRIR ELE. OS FORMATOS RAR, 7Z E OUTROS NÃO SÃO SUPORTADOS NATIVAMENTE, AÍ OS USUÁRIOS PRECISAM FAZER A INSTALAÇÃO MANUAL... QUE FICOU MAIS DIFÍCIL NO ANDROID 11.**

## A estrutura de arquivos do mod de mapa

O mapa no Proton é composto de vários arquivos txt (de texto puro), e algumas pastas. A estrutura primária essencial é um txt de definição do mapa e uma pasta, ambos ficando dentro da pasta “maps”, que por sua vez fica na pasta de mods do jogo. A pasta de mods normalmente é a Proton Bus Mods dentro da documentos, na versão PC; e a pasta com.viamep.p.../files, dentro da Android - data, no celular.

Exemplo da estrutura no PC:



O txt de definição do mapa deve terminar em **.map.txt**. O jogo procura todos os arquivos **.map.txt** nesta pasta para listar no menu de seleção. A partir do arquivo selecionado ele lerá as definições e então puxará os arquivos dos modelos, texturas, etc. Normalmente você usará o nome do mapa ou a região, linha etc, como no exemplo acima, **Estrada Longa.map.txt**. Deve ser um nome único, que outras pessoas não utilizem! Afinal, se utilizarem, as pessoas estariam sobrescrevendo seu mapa ao instalar. Evite nomes comuns. Uma dica é usar um prefixo ou sufixo só seu, como o nome do autor ou equipe junto, por exemplo.

**IMPORTANTE: JAMAIS USE ACENTOS, CE-CEDILHA OU CARACTERES ESPECIAIS NOS NOMES DE ARQUIVOS DOS MODS! ELES PODEM FUNCIONAR NO SEU COMPUTADOR, MAS PODERÃO FALHAR EM SISTEMAS EM OUTROS IDIOMAS.**

Dentro deste txt teremos uma estrutura básica parecida com esta:

**[map]**

**baseDir=Estrada Longe** << A pasta base do mapa, que fica na maps junto ao .map.txt

**modelsDir=Rota 1** << A pasta dos modelos, que fica dentro da pasta **tiles**, esta por sua vez deve ficar dentro da pasta base definida acima

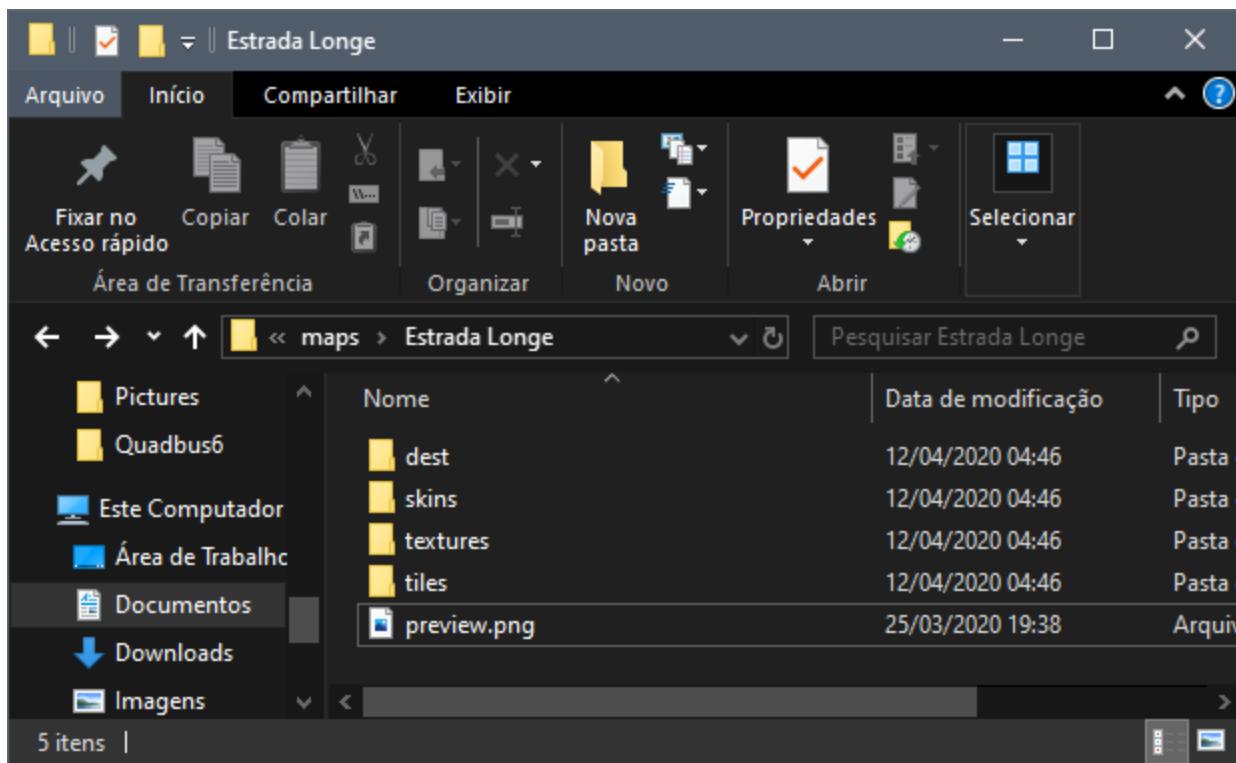
**textures=textures** << A pasta das texturas, que fica dentro da pasta base

**mapModVersion=3** << A versão do sistema do mod, que informa ao jogo para emitir uma mensagem caso a pessoa tente usar mods do futuro em versões antigas

**IMPORTANTE: NA FASE 2 UTILIZE O NÚMERO 2 NESTE ITEM!**

**preview=preview.png** << A imagem que será exibida na tela de seleção do mapa. Ela deve ficar dentro da pasta base, não nas texturas. Se esse item for omitido, ele buscará por uma imagem chamada “preview.png”. Recomendamos a proporção 16:9 para esta imagem. Um tamanho bom é 640x360 pixels. Não precisa ser muito grande.

Dentro da pasta base do mapa devem existir outras pastas, como estas:



A pasta **dest** armazenará os letreiros e placas das linhas. Ela terá subpastas com os destinos definidos mais adiante, e dentro de cada pasta de destino, as imagens referentes a eles.

A pasta **skins**, como o nome sugere, armazenará as pinturas dos ônibus do mapa. Deve existir uma pasta chamada **0** (zero) dentro dela, e dentro da **0** as pastas dos ônibus (só o **pbc**, no caso). As skins são sorteadas aleatoriamente entre as que existirem dentro da pasta do ônibus.

A pasta **textures**, como o nome sugere, armazena as texturas dos objetos do mapa.

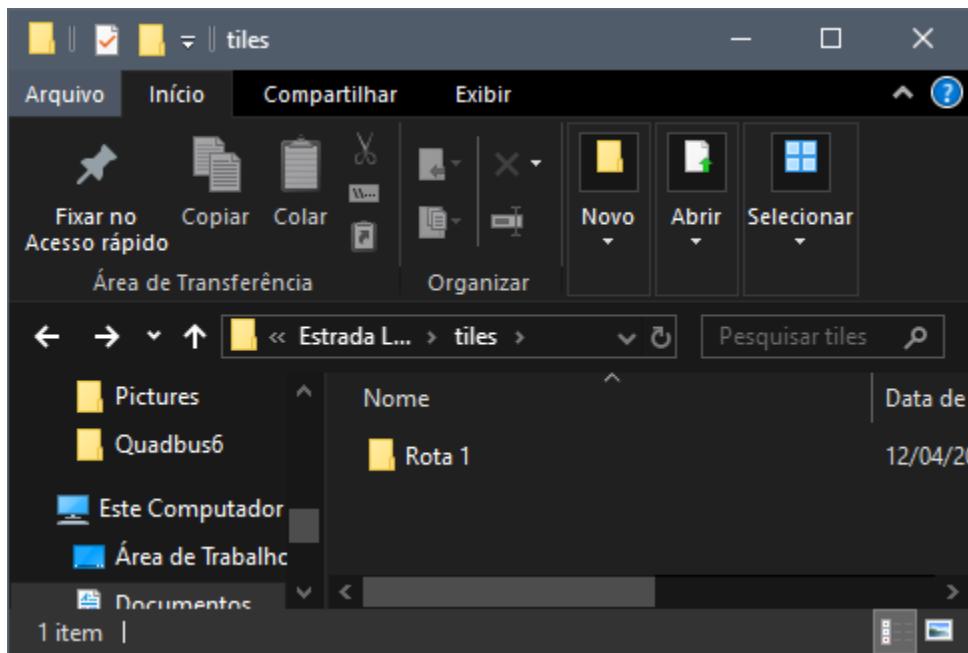
**IMPORTANTE: EVITE TEXTURAS EM JPG! APESAR DO JOGO TENTAR CARREGÁ-LAS, ELE FUNCIONA MELHOR COM PNG. ALGUMAS JPG PODEM FICAR ROXAS OU NÃO SEREM CARREGADAS NO JOGO, ESPECIALMENTE NOS CELULARES.**

A pasta **tiles** deve ter este nome, obrigatoriamente. Dentro dela ficará a pasta dos modelos, aquela definida no **.map.txt** em **modelsDir**. Essa estrutura foi feita pensando num eventual sistema de carregamento contínuo no futuro, apesar de não ser garantido, apenas um estudo.

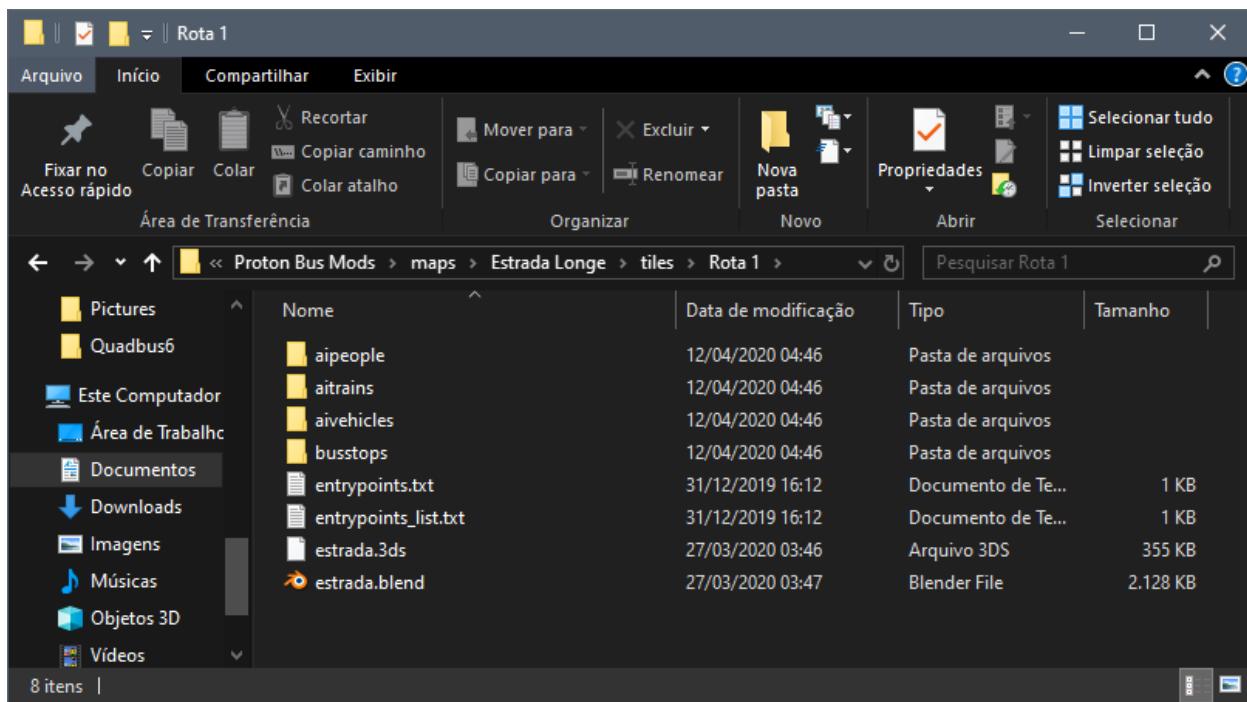
Ele será evitado pois pode fazer o jogo travar constantemente ao carregar cenário, não queremos isso.

**IMPORTANTE: OS MODS DE MAPAS DO PROTON BUS SÃO CARREGADOS POR INTEIRO! NÃO PODEM SER MUITO PESADOS, COM MUITAS LINHAS. CASO VÁ FAZER VÁRIAS LINHAS LONGAS, DIVIDA ELAS EM PARTES, SEPARANDO EM VÁRIOS “MAPAS” TEMÁTICOS DEDICADOS A ELAS. CADA UMA COM SEU .MAP.TXT, DE FORMA QUE O USUÁRIO SÓ CARREGUE A QUE FOR USAR. ISSO IMPEDE LIGAÇÕES CONTÍNUAS, MAS FICA MAIS LEVE, ESPECIALMENTE CONSIDERANDO OS CELULARES...**

No nosso exemplo, dentro da pasta tiles tem a pasta **Rota 1**, que foi definida no modelsDir lá no txt principal do mapa.



Dentro desta pasta dos modelos é que vão ficar os arquivos exportados (3ds, no caso), além de dois txts referentes aos locais de posicionamento ou entrada do ônibus, e outras pastas específicas:



Todos os cenários do mapa em questão devem ser exportados para .3ds no Blender e colocados nesta pasta. Você pode dividir seu modelo em camadas ou arquivos diferentes, e colocá-los todos nesta pasta.

**DICA: SALVE O BLEND DIRETO NESSA PASTA, ASSIM NA HORA DE EXPORTAR FICA MAIS FÁCIL! APAGUE O BLEND AO REDISTRIBUIR SEU MAPA, CASO VOCÊ NÃO QUEIRA LIBERÁ-LO PARA EDIÇÕES.**

**FALANDO NISSO, MUITO IMPORTANTE: OS MODS DO PROTON BUS NÃO SÃO PROTEGIDOS! TECNICAMENTE, NENHUM JOGO É! SE APARECER NA TELA, HÁ PROGRAMAS QUE PODEM CAPTURAR O 3D DIRETO DA MEMÓRIA DA PLACA DE VÍDEO. POR ISSO NÃO SERÁ FEITO NENHUM SISTEMA DE PROTEÇÃO, SERIA UM GASTO DE ENERGIA E ESFORÇOS INÚTEIS. A ÚNICA FORMA DE NÃO TER SEU MOD EDITADO INDEVIDAMENTE OU VAZADO É NÃO LANÇANDO ELE.**

O conteúdo desta pasta, além dos arquivos 3ds e do blend, se resume a estes itens:

<b>aippeople</b>	<< Pasta com as configurações dos pedestres, pessoas andantes
<b>aitrains</b>	<< Pasta com as configurações dos trens
<b>aivehicles</b>	<< Pasta com as configurações dos carros do tráfego
<b>busstops</b>	<< Pasta com as configurações dos pontos de ônibus e passageiros
<b>entrypoints.txt</b>	<< Configurações avançadas dos pontos de entrada e posicionamento
<b>entrypoints_list.txt</b>	<< Lista simples dos nomes dos pontos de entrada e posicionamento

Os itens de configuração destas pastas estão diretamente relacionados a peças devidamente nomeadas nos arquivos 3D do mapa. O jogo irá ler estes arquivos e buscar pelas peças com os nomes sugeridos na hora de ligar as informações. Por exemplo, dentro da pasta aivehicles ficam arquivos txt com informações sobre os paths de caminhos dos carros. Lá cada rua ou avenida, ou segmento, terá um nome. O jogo vai ler este nome e procurar nos modelos do mapa carregado onde eles estão, para então montar o código que fará os carros aparecerem nestes lugares. É de sua responsabilidade manter um sistema de nomes legível e único, que não gere confusão na hora de configurar. Veremos algumas dicas logo mais.

## Configurando os pontos de entrada

Logo após selecionar o mapa no jogo os usuários precisam selecionar a rota, linha, destino ou local de entrada (pode ser uma garagem, alguma rua etc, não necessariamente uma linha). Este ponto de entrada será identificado como um **entrypoint**. Você pode ter quantos quiser dentro do mapa.

**IMPORTANTE: LEMBRE-SE DA RECOMENDAÇÃO DE NÃO FAZER MAPAS COM MUITAS LINHAS, FOCO MAIS NOS CENÁRIOS ONDE O JOGADOR VAI PASSAR! SENÃO PODE FICAR PESADO PARA OS CELULARES, POR TER TUDO CARREGADO DE UMA VEZ.**

Os pontos de entrada devem ser nomeados de forma única, de forma que nenhum outro objeto no mapa os tenha, para evitar que o código bugue encontrando o objeto errado. Uma sugestão é colocar algo que identifique a linha e o sentido, por exemplo, como nos mapas nativos:

**351F-10 TP** << O ponto de entrada inicial da linha, terminal primário

**351F-10 TS** << O ponto de entrada final da linha, terminal secundário

**Garagem Maria Amalia** << Um nome de garagem

**IMPORTANTE: JAMAIS USE ACENTOS OU CARACTERES ESPECIAIS NESTES NOMES! ISSO EVITA PROBLEMAS DIVERSOS AO USAR O JOGO EM SISTEMAS DE OUTROS IDIOMAS. NÃO PODE USAR A BARRA TAMBÉM, POIS ELA É SEPARADOR DE ARQUIVOS NOS SISTEMAS OPERACIONAIS, E ESTES NOMES SERÃO USADOS EM ARQUIVOS.**

**USE APENAS LETRAS DE A A Z, NÚMEROS DE 0 A 9 OU UM TRAÇO SIMPLES OU SUBLINHADO (UNDERLINE).**

Se você não quiser seguir o estilo São Paulo, tudo bem, pode usar algo como no mapa Uipe:

**77-01** << A linha 77, rota 1, ida

**77-02** << A linha 77, roda 2, volta

Não pode usar nomes repetidos, como só “77”. Todavia, pode usar “77 ida” e “77 volta”, normalmente. Estes nomes serão usados nas pastas dos letreiros.

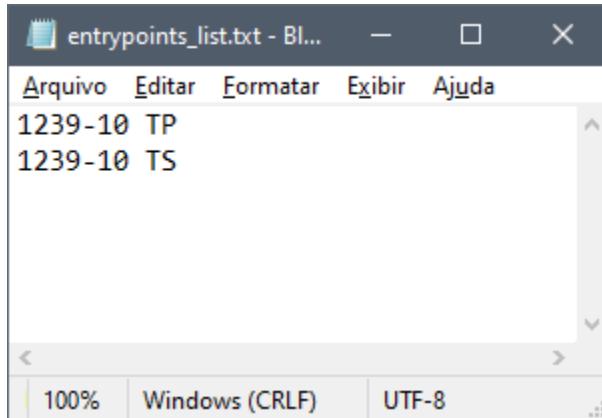
**IMPORTANTE: NENHUM OUTRO OBJETO DO MAPA PODERÁ TER ESTES NOMES! SENÃO O JOGO PODERÁ SELECIONAR ELES POR ENGANO AO TENTAR ATRIBUIR O LOCAL FÍSICO DA ENTRADA. POR ISSO É BOM USAR TP OU TS, OU -01 E -02, OU IDA E VOLTA, ETC.**

Não use, por exemplo, 001, Cube, Rua, etc. Precisa ser um nome único específico para o ponto de entrada.

**NOVAMENTE, REAFIRMAMOS: NÃO USE CE-CEDILHA, ACENTOS, ASPAS, CARACTERES DIFERENCIADOS DE NENHUMA ESPÉCIE NESTES NOMES! MESMO QUE TALVEZ FUNCIONE PARA VOCÊ, NÃO SÃO OFICIALMENTE SUPORTADOS E PODEM BUGAR O SISTEMA.**

Enfim, definidos os nomes, vamos aos arquivos!

O arquivo **entrypoints\_list.txt** deve conter uma lista simples dos nomes em questão, um em cada linha. Ele será usado pelo jogo ao mostrar a lista de linhas no menu de seleção para o jogador, bem como na lista de reposicionar o ônibus.



Já o arquivo **entrypoints.txt** é um pouco mais complexo. Ele marcará, de fato, o local físico no mapa onde o ônibus será posicionado.

```

[entrypoint_1]
name=1239-10 TP
posX=-40.9988
posY=0.086769
posZ=49.1041
rotX=
rotY=180
rotZ=

[entrypoint_2]
name=1239-10 TS
posX=-24.1497
posY=0.252659
posZ=-5526.82
rotX=
rotY=
rotZ=

```

IMPORTANTE: LEMBRE-SE QUE NO BLENDER E NA UNITY OS EIXOS Y E Z SÃO INVERTIDOS! NO BLENDER O Y É PARA FREnte/TRÁS E O Z PARA CIMA/BAIXO; NA UNITY O Y É PARA CIMA/BAIXO E O Z PARA FREnte/TRÁS! SEMPRE AO COPIAR UMA COORDENADA NO Y DO BLENDER, COLE NO Z DA UNITY, E VICE-VERSA. O X NÃO MUDA!

O **entrypoints.txt** terá uma lista com várias seções incrementais do tipo entrypoint\_1, entrypoint\_2, entrypoint\_3 etc... Quantos forem os pontos de entrada necessários. No mapa de exemplo da imagem, são só dois.

IMPORTANTE: SE VOCÊ ADICIONAR OU REMOVER NOVOS PONTOS DE ENTRADA POSTERIORMENTE, ATUALIZE ESTE ARQUIVO, MANTENDO O NÚMERO DE FORMA INCREMENTAL, INICIANDO EM 1.

A estrutura do entrypoint aqui é bem direta:

**[entrypoint\_1]** << O identificador do entrypoint nessa lista

**name=1239-10 TP** << O nome do entrypoint em questão, o mesmo colocado no entrypoints\_list.txt e também será o mesmo da pasta dos letreiros

**posX=-40.9988** << A posição do entrypoint no eixo X no modelo 3D

**posY=0.086769** << A posição do entrypoint no eixo Y no modelo 3D (Z do Blender, é a altura!)

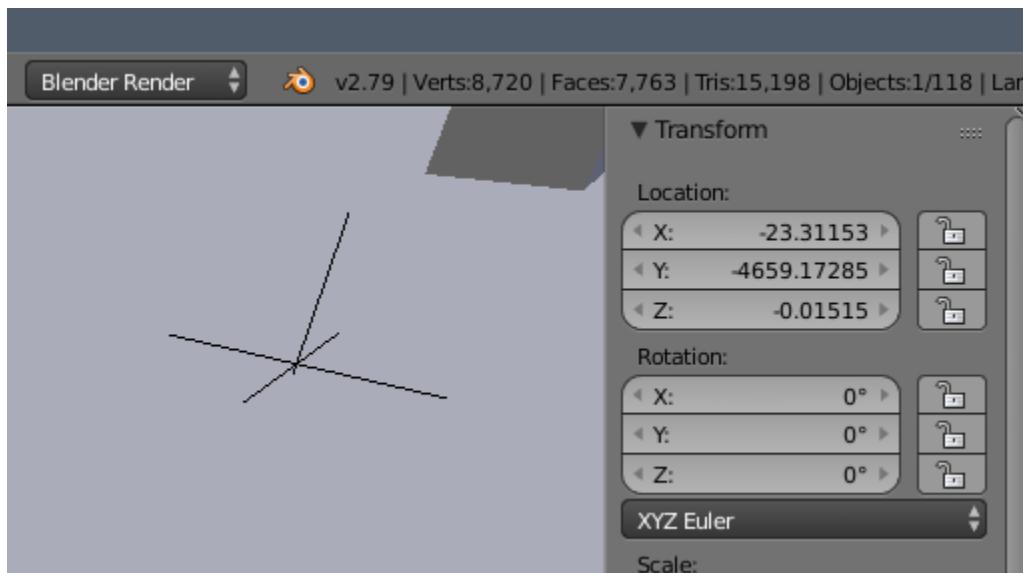
**posZ=49.1041** << A posição do entrypoint no eixo Z no modelo 3D (Y no Blender!)

**rotX=** << A rotação em relação ao X, normalmente não precisa, será sempre 0

**rotY=180** << A rotação em relação ao eixo Y (Z no Blender), indicando para onde o ônibus estará apontado!

**rotZ=** << A rotação em relação ao eixo Z (Y no Blender), normalmente não precisa, será sempre 0

Para saber quais são estas coordenadas, você poderá usar um objeto empty no Blender, é uma das formas mais fáceis. Por exemplo:



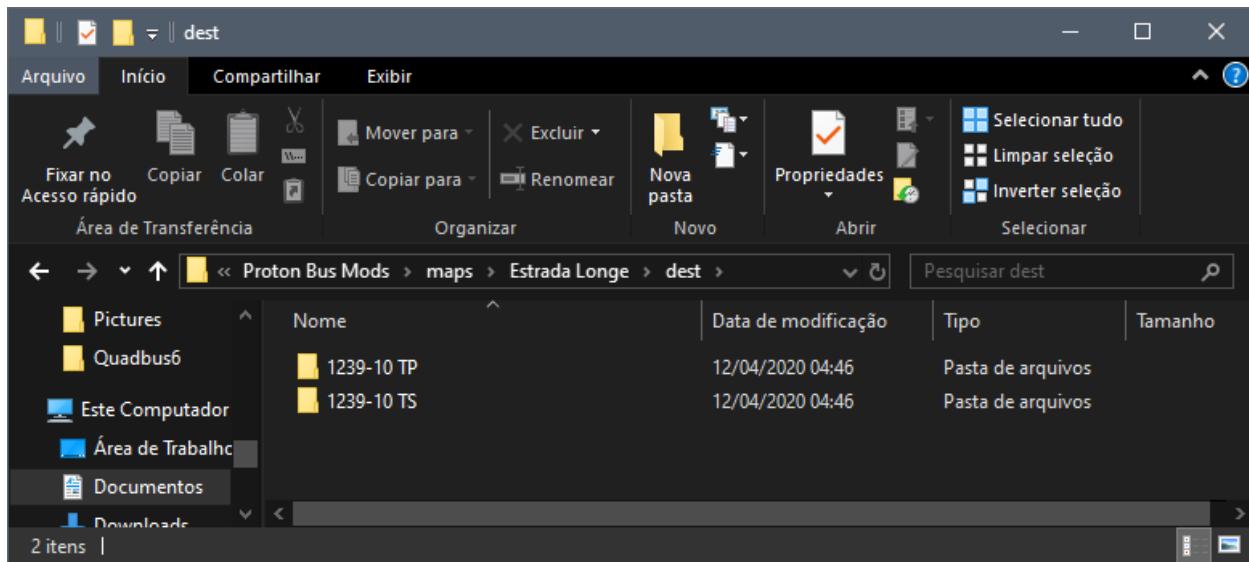
**IMPORTANTE: SEMPRE USE O PONTO FINAL COMO SEPARADOR DECIMAL, NÃO A VÍRGULA! O JOGO TENTARÁ LER O PONTO, SE COLOCAR VÍRGULA, PODERÁ DAR ERRO E TRAVAR DALI ATÉ O FINAL.**

A rotação no eixo vertical (Z no Blender, Y no jogo) é importante para definir a orientação, se estará “reto” ou virado 90 graus, -90, 180, 15, 32 etc.

**DICA:** esta coordenada precisa ficar levemente acima do piso, não dentro dele, senão o ônibus pode cair no limbo infinito ao ser posicionado! Não precisa ser muito alto, mas alguns cm do chão devem ajudar. O ponto 0,0,0 do ônibus coincidirá com este lugar quando ele for adicionado ou reposicionado no mapa.

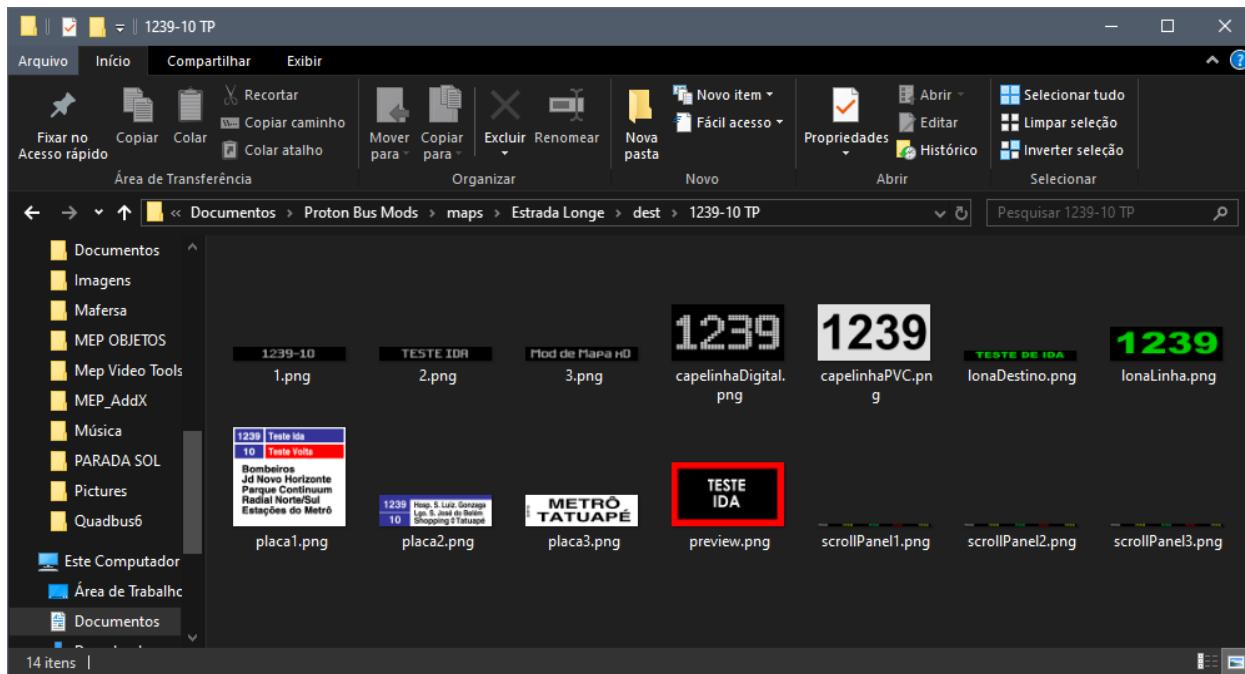
Lá na pasta base do mapa deve ter uma pasta chamada **dest**, e dentro dela terão pastas com os nomes dos entrypoints, exatamente como definidos no **entrypoints\_list.txt** e na propriedade **name** no **entrypoints.txt**.

**IMPORTANTE:** É obrigatório manter consistência nos nomes! Se tiver um caractere diferente ou espaço a mais ou a menos, poderá não funcionar depois.



Por isso não é permitido usar barras nos entrypoints: eles são usados no sistema de arquivos, e o Windows e outros sistemas operacionais não permitem usar barras em nomes de arquivos e pastas por ser o caractere separador de diretório. Ao encontrar uma barra o sistema a trata como uma divisão de pasta, não como parte do nome do arquivo.

Dentro da pasta do destino ficarão os arquivos das imagens dos destinos e placas, além de uma imagem **preview.png**, que será usada na tela de seleção da rota no jogo.



**IMPORTANTE: EVITE IMAGENS MAIORES DO QUE 2048 PIXELS DE LARGURA OU ALTURA PARA OS MODS NOS CELULARES! A MAIORIA DAS GPUS MOBILE NÃO SUPORTAM ESTAS IMAGENS, PODENDO CAUSAR CRASHES OU INSTABILIDADES DIVERSAS NO JOGO.**

Os nomes e tamanhos sugeridos são os mesmos para os mods de letreiros dos jogadores, no caso, estes:

**1.png** Letreiro principal frontal, eletrônico, fase 1 (1024x128)

**2.png** Letreiro principal frontal, eletrônico, fase 2 (1024x128)

<b>3.png</b>	Leteiro principal frontal, eletrônico, fase 3 (1024x128)
<b>capelinhaDigital.png</b>	Leteiro eletrônico traseiro ou lateral com o número da linha (256x128)
<b>capelinhaPVC.png</b>	Placa de plástico ou papel com o número da linha atrás (256x128)
<b>lonaDestino.png</b>	Parte do letreiro de lona dianteiro com a linha (256x128)
<b>lonaLinha.png</b>	Parte do letreiro de lona dianteiro com o destino (1024x128)
<b>placa1.png</b>	Placa lateral (400x350)
<b>placa2.png</b>	Placa frontal de apoio (512x142)
<b>placa3.png</b>	Placa frontal de indicação de vias (512x142)
<b>scrollPanel1.png</b>	Leteiro rolante principal, em alguns ônibus (2048x32)
<b>scrollPanel2.png</b>	Leteiro rolante auxiliar, opcional (2048x32)
<b>scrollPanel3.png</b>	Leteiro rolante auxiliar, opcional (2048x32)

Os tamanhos são apenas sugestões para manter a proporção, mas podem ser maiores ou menores. Apenas realmente evite acima de 2048.

Além destes, deve ter um **preview.png** também, que será exibido na tela de seleção da linha no jogo. Recomendamos a proporção 16:9 para esta imagem de preview. Um tamanho bom é 640x360 pixels. Não precisa ser muito grande.

## Definição da característica da rota: rodoviária ou urbana?

Novo a partir da v253: se a rota em questão for uma rota rodoviária onde as pessoas só devem desembarcar no ponto final, e não em pontos intermediários, coloque um txt vazio na pasta dos letreiros com o nome **intercity.txt**. As rotas que tiverem este txt na pasta do letreiro vão fazer as pessoas só descerem quando passarem pelo trigger de desembarque, idealmente no último ponto da linha. Sem esse txt a rota será tratada normalmente como urbana, onde as pessoas

pedem para descer depois de alguns minutos que estiverem no ônibus (o tempo é bastante aleatório para cada pessoa).

## É letreiro reservado, garagem ou algum que proíba a entrada?

Novo a partir da v253: se for um letreiro que não permita o embarque de passageiros, coloque um txt vazio na pasta do destino chamado **outofservice.txt**. Isso fará com que as pessoas não embarquem. É ideal para letreiros como GARAGEM, RESERVADO, etc.

**IMPORTANTE: OS NOMES SÃO CASE SENSITIVE! É NECESSÁRIO DIFERENCIAR MAIÚSCULAS DE MINÚSCULAS PARA QUE FUNCIONE EM TODOS OS SISTEMAS OPERACIONAIS. SEM RESPEITAR ISSO TALVEZ ATÉ FUNCIONE DE BOA NO SEU SISTEMA, MAS NÃO EM OUTROS, POIS HÁ SISTEMAS QUE CONSIDERAM A DIFERENCIACÃO.**

Isso é tudo sobre os entrypoints por enquanto. Futuramente eles terão alguma configuração relacionada ao GPS para exibição, mas isto não foi implementado na fase 2 dos mods.

## Configurando os passageiros

Atualmente no Proton as pessoas entram em qualquer ponto, independente do letreiro configurado. Futuramente isso poderá ser alterado para ficar mais real, evitando que embarquem com o destino de outra linha.

A configuração dos pontos é dividida em duas partes: alguns objetos no modelo 3D e alguns parâmetros num arquivo txt. Os objetos vão definir o local do ponto e onde as pessoas ficam esperando, e o txt definirá algumas propriedades como o nome do ponto, quantidade de passageiros a embarcar ali, a rotação deles (para onde estarão apontados), etc.

**IMPORTANTE: CADA PONTO DE ÔNIBUS NO PROTON DEVE TER UM NOME ÚNICO, EXCLUSIVO, QUE NÃO É USADO POR NENHUM OUTRO MODELO DO MAPA! NO CARREGAMENTO SERÁ FEITA UMA BUSCA PELOS NOMES DOS OBJETOS. REUTILIZAR NOMES COMUNS A OUTRAS PEÇAS PODEM CAUSAR PROBLEMAS, JÁ QUE O JOGO PODE ENCONTRAR O OBJETO ERRADO. PARA EVITAR ISSO É BOM CRIAR UM PADRÃO PRÓPRIO DE NOMENCLATURA, SEMPRE USANDO ALGUM PREFIXO CARACTERÍSTICO DOS PONTOS, EVITANDO CONFUSÃO. O MESMO VALERÁ PARA O SISTEMA DE TRÁFEGO, VISTO MAIS ADIANTE. NÃO USE ACENTOS OU CARACTERES ESPECIAIS NOS NOMES DE OBJETOS LIDOS PELO JOGO, VALE A MESMA REGRA PARA OS ARQUIVOS!**

É recomendável que o nome interno seja uma palavra única, sem espaços, com algum prefixo que identifique que é um ponto, seguido de um nome exclusivo para este ponto. Por exemplo, algumas sugestões de nomes de pontos:

zzPonto1, zzPonto2, zzPontoDaPraca, zzPontoRuaXavier1, zzPontoRuaXavier45...

Exemplos de nomes ruins, que poderiam causar problemas com outros objetos do mapa:

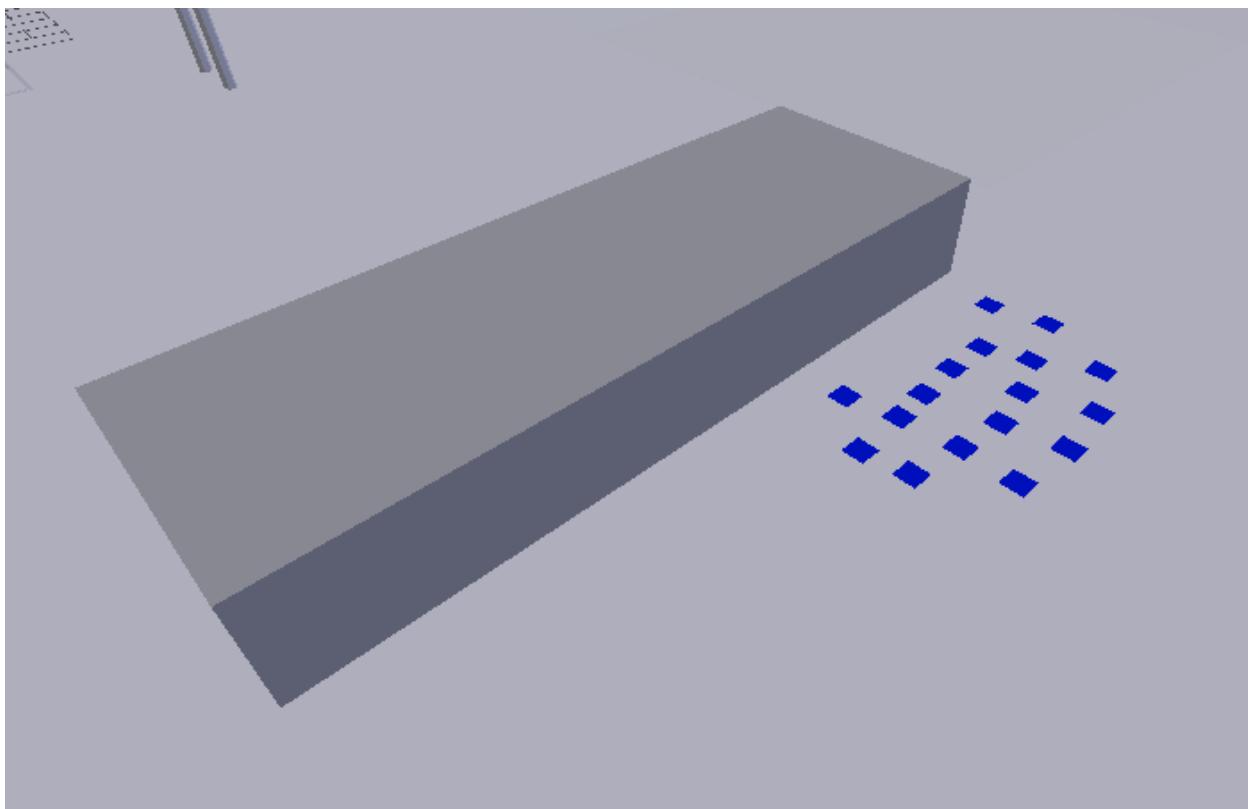
rua, cube, Cube, collider, stop, bus...

Não é obrigatório usar "zz", é só uma sugestão. Fica fácil de procurar no Blender depois, caso queira apagar todos os objetos do ponto. Nomes sem nenhum prefixo podem ficar mais difíceis de encontrar em mapas grandes.

No modelo 3D o ponto terá dois tipos de objetos: uma forma cúbica onde o ônibus deve estar com um pedaço dentro, para identificar que está nessa parada; e alguns objetos pequenos no chão, que serão as posições dos passageiros em pé esperando.

**OBSERVAÇÃO: NO MOMENTO O SISTEMA NÃO SUPORTA PASSAGEIROS SENTADOS NO PONTO, SOMENTE EM PÉ!**

### A configuração dos pontos do modelo 3D



O cubo, ou melhor, paralelepípedo, deve ser criado a partir de um cubo, sem muitos vértices desnecessários. Ele será um trigger no jogo, um colisor invisível que identifica quando o ônibus

está dentro dele. Os pontinhos pequenos são objetos para identificar as posições dos passageiros aguardando.

**DICA: PARA DEIXAR O MAPA MAIS LEVE, UTILIZE PLANOS SIMPLES EM VEZ DE CUBINHOS NOS OBJETOS PEQUENOS DAS POSIÇÕES DAS PESSOAS. OS PLANOS POSSUEM APENAS DOIS TRIÂNGULOS, ENQUANTO QUE OS CUBOS POSSUEM DOZE (2 TRIÂNGULOS POR FACE, 6 FACES). ESTES OBJETOS SERÃO DESCARTADOS NO CARREGAMENTO, NÃO PRECISAM SER PESADOS, SERVEM APENAS PARA PEGAR AS POSIÇÕES. NÃO FUNCIONA COM OS OBJETOS EMPTY DO BLENDER, PRECISA TER UMA MALHA, ENTÃO QUE ELA SEJA O MAIS SIMPLES POSSÍVEL!**

O nome do colisor invisível deve ser **nomedoponto\_trigger**. As posições de passageiros devem ter o nome do ponto seguido de um ponto final, e números incrementais a partir do 000. Isso foi definido como uma conveniência, já que utilizamos o Blender. Basta duplicar o primeiro objeto nomeado com o final .000 que ele vira .001, .002, .003 etc. Isto será adotado também nos paths de tráfego.

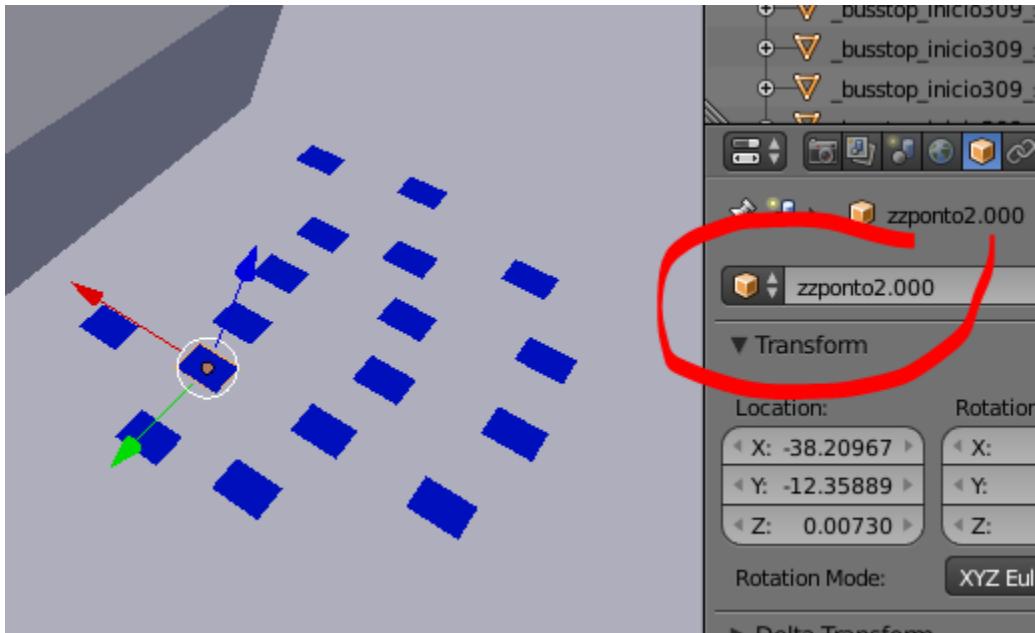
Por exemplo, como ficaria:

Para o ponto de nome interno *zzPonto1*:

*zzPonto1\_trigger, zzPonto1.000, zzPonto1.001, zzPonto1.002, zzPonto1.003 etc*

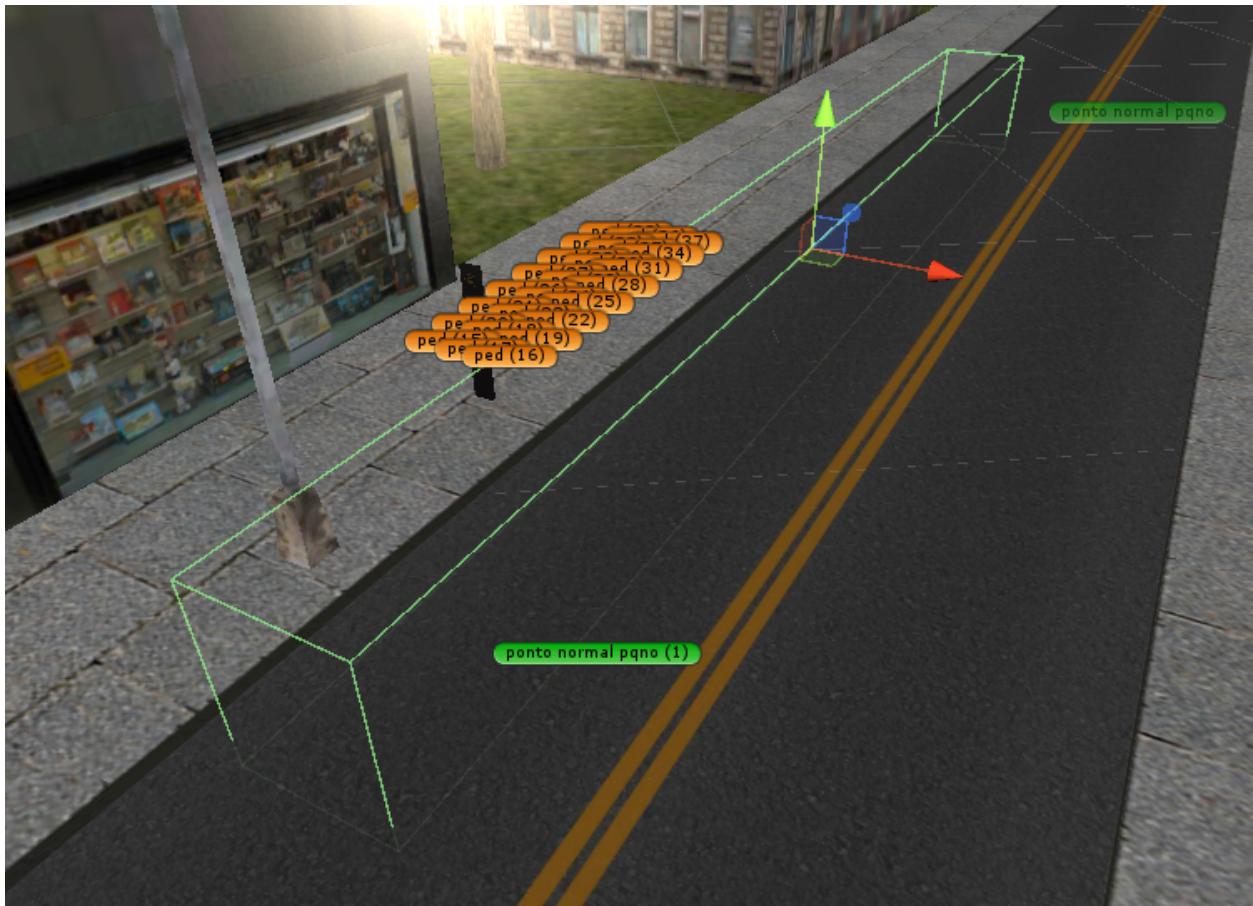
Para algum ponto chamado *zzAvenidaContinuum509*:

*zzAvenidaContinuum509\_trigger, zzAvenidaContinuum509.000, zzAvenidaContinuum509.001, zzAvenidaContinuum509.002 etc*

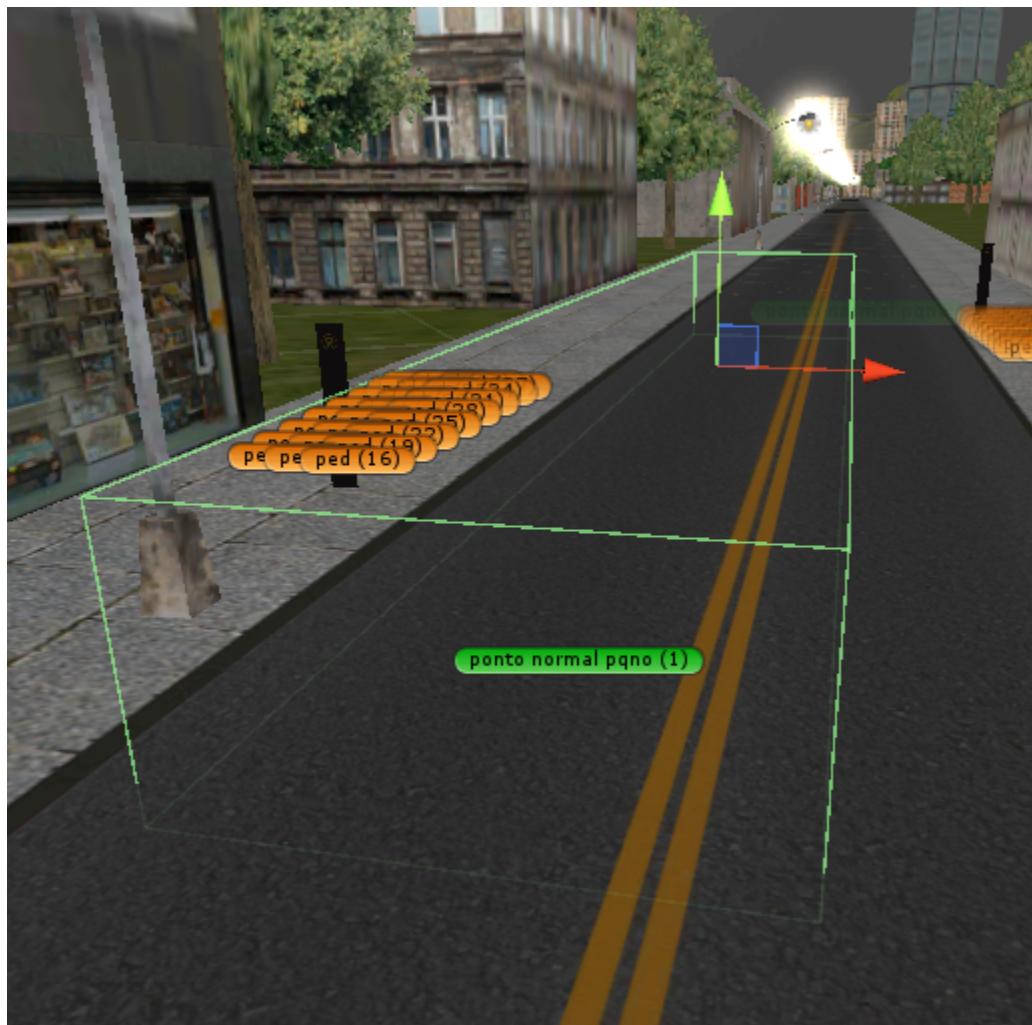


IMPORTANTE: O COLISOR DO PONTO DEVE PEGAR UMA ÁREA BOA DA BAIA DELE, PORÉM NÃO PODE VARAR PARA O OUTRO LADO DA RUA, CASO SEJA UMA VIA DE DUAS FAIXAS EM SENTIDOS OPOSTOS. É BOM DEIXAR UM PEDAÇO PRA FORA NA RUA NO MESMO SENTIDO EM QUE O ÔNIBUS FOR PASSAR, PERMITINDO QUE O JOGO IDENTIFIQUE QUANDO ALGUÉM QUER DESCER PARA EMITIR A RECLAMAÇÃO CASO O MOTORISTA NÃO PARE. O VEÍCULO PRECISARÁ ESTAR COM PELO MENOS UM PEDAÇO DENTRO DO TRIGGER PARA QUE OS PASSAGEIROS POSSAM ENTRAR.

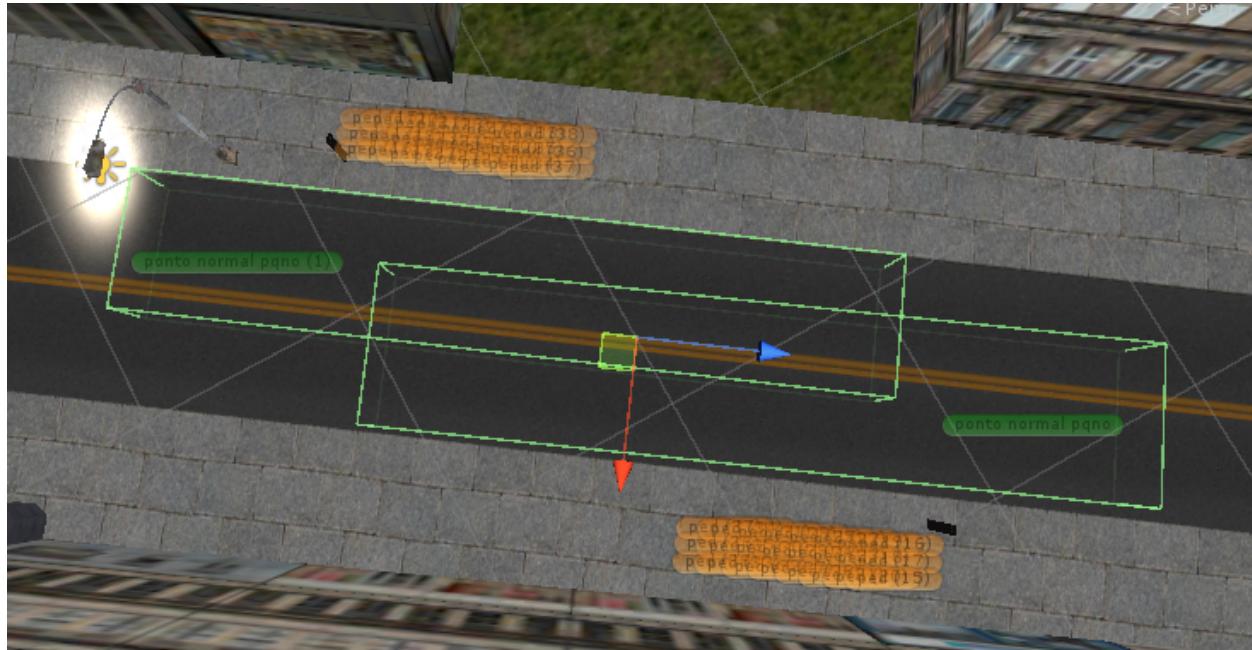
Exemplo de trigger bem posicionado:



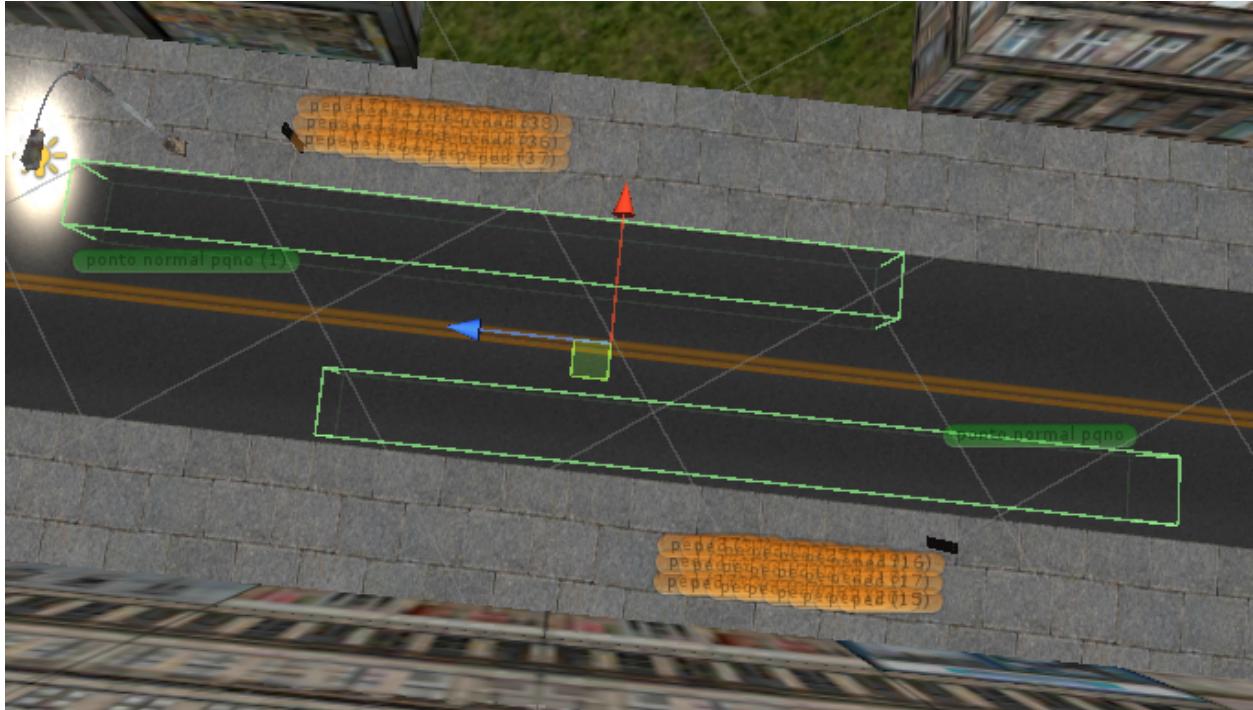
Exemplo de trigger mal posicionado, identificando o ônibus no ponto mesmo no sentido oposto:



Exemplo de triggers mal posicionados, com os dois pontos sobrepostos, não pode:



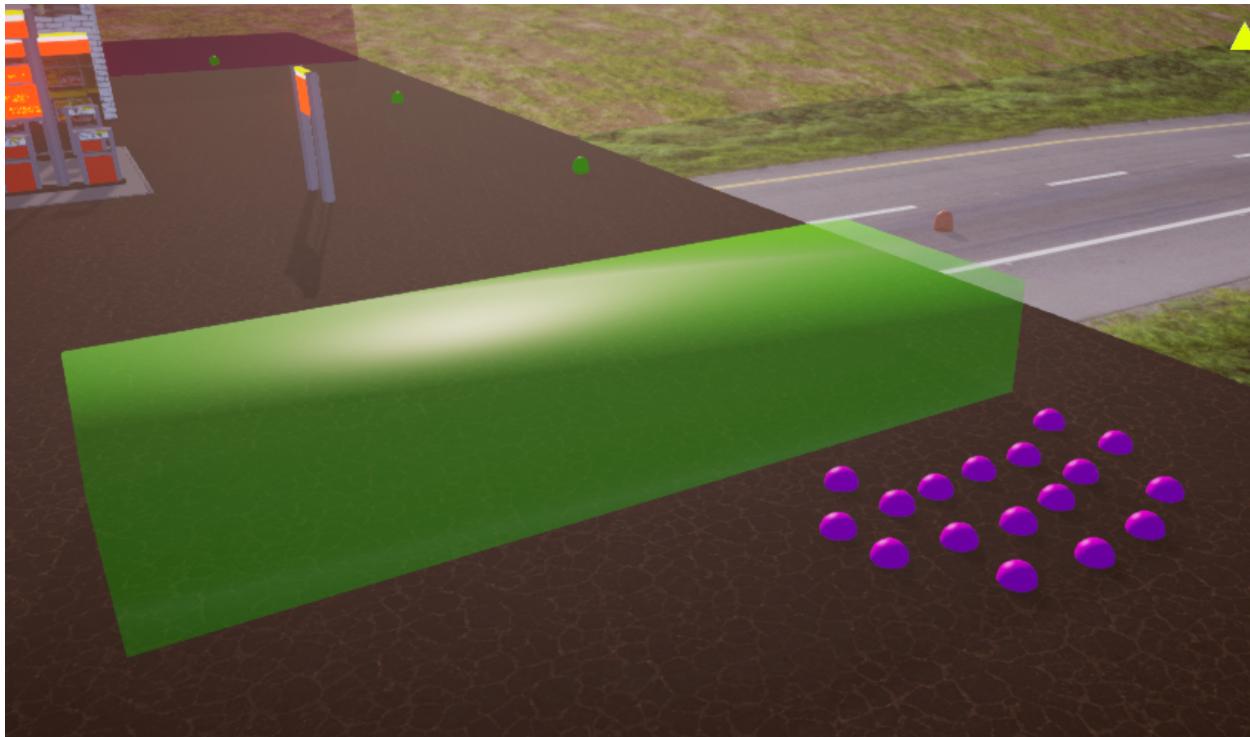
Exemplo de pontos opostos corretos, bem posicionados:



Note que o ônibus não pode estar sobre dois triggers de ponto ao mesmo tempo, caso contrário resultados inesperados podem ocorrer, com as pessoas vindo do outro lado e passando por cima, por exemplo.

DICA: ATIVE AS OPÇÕES DE DEBUG NO MAPA NO JOGO PARA VISUALIZAR ONDE ESTÃO OS PONTOS, TRIGGERS E OBJETOS DE POSIÇÃO. ISSO PERMITE IDENTIFICAR ALGUNS ERROS QUE PODEM PASSAR DESPERCEBIDOS DENTRO DO BLENDER.

Exemplo de visualização no modo debug:



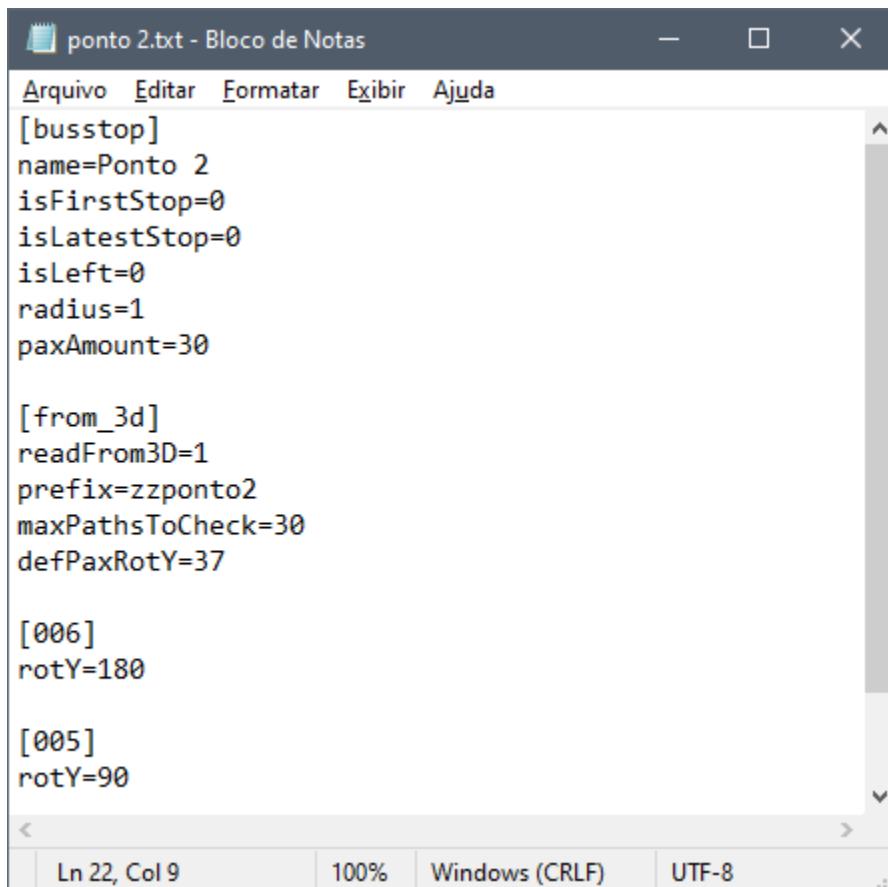
DICA: NÃO PRECISA COLOCAR MATERIAL NEM TEXTURA NO TRIGGER NEM NOS OBJETOS DE POSIÇÃO. COMO ESTAS PEÇAS VÃO SER DESCARTADAS E SÓ SERVEM PARA PEGAR AS POSIÇÕES, É MELHOR DEIXAR SEM MATERIAL PARA FICAR MAIS LEVE O PROCESSO DE CARREGAMENTO.

## A configuração dos pontos no txt

Para cada ponto deve ter um txt dentro da pasta **busstops**, que fica dentro da **modelsDir** definida no txt do mapa. Se você colocar o ponto no modelo 3D e esquecer do txt, ou se tiver alguma inconsistência nos nomes, talvez o ponto não funcione no jogo ou apareça como um objeto rosa/sem material.

Uma boa prática é nomear o txt com o mesmo nome do ponto, aquele nome usado no trigger e nos objetos das posições. Apesar de isso não ser obrigatório nas versões atuais é bom deixar assim, fica mais organizado.

O conteúdo do txt é mais ou menos isso:



```
[busstop]
name=Ponto 2
isFirstStop=0
isLatestStop=0
isLeft=0
radius=1
paxAmount=30

[from_3d]
readFrom3D=1
prefix=zzponto2
maxPathsToCheck=30
defPaxRotY=37

[006]
rotY=180

[005]
rotY=90
```

A primeira seção **[busstop]** define algumas propriedades básicas do ponto.

O item **name** define um nome amigável do ponto, que eventualmente poderá aparecer para o jogador no futuro. Normalmente pode ser um nome com espaços, algo que seria aplicado em uma futura lista de linhas. No momento ele não será usado, mas se quiser já pode deixar definido. Vale a mesma regra: evite acentos ou caracteres especiais.

As propriedades **isFirstStop** e **isLatestStop** provavelmente serão abandonadas em versões futuras do jogo por algo melhor. Em tese, você colocaria o valor 0 para false/negativo e 1 para true/positivo.

**isFirstStop** seria para definir que este é o primeiro ponto da linha, orientando ao sistema do jogo de que as pessoas não tentem desembarcar ao permanecer com o ônibus ali (enquanto o motorista enrola parado falando numa gameplay, por exemplo). E **isLatestStop** é para o último, orientando todo mundo a desembarcar. Todavia o desembarque para esvaziar o veículo é gerido por um trigger à parte, esta configuração **isLatestStop** atualmente é obsoleta.

**AVISO: EM VERSÕES FUTURAS OS PONTOS INICIAL E FINAL SERÃO IDENTIFICADOS A PARTIR DE UMA LISTA DE PONTOS PARA CADA LINHA, DISPENSANDO O USO DESTA CONFIGURAÇÃO. POR ENQUANTO ELAS PODEM SER UTILIZADAS, MAS NÃO SÃO LÁ TÃO IMPORTANTES. NOS PONTOS INTERMEDIÁRIOS, QUE NÃO SÃO INICIAIS NEM FINAIS, DEIXE AMBOS COM O VALOR EM ZERO. VOCÊ TAMBÉM PODE OMITIR ESTES ITENS NO TXT QUE ELES ASSUMIRÃO AUTOMATICAMENTE O VALOR PADRÃO (ZERO).**

**isLeft** indica ser uma parada à esquerda, independente de ser a nível da calçada ou em plataforma elevada. O padrão é 0 (false), indicando um ponto à direita. Utilize **isLeft=1** para pontos à esquerda, como em corredores ou BRTs. É possível mesclar pontos à esquerda e direita numa mesma rota. Ao entrar no trigger do ponto do lado oposto os passageiros tentam mudar de lado.

A propriedade **radius** não será usada nesta forma de configuração, ela foi feita durante o desenvolvimento do beta interno com colisores gerados de outra forma. Basta ignorar ou deixar sempre no valor de 1 mesmo, não é mais necessária.

Por fim, **paxAmount** define a quantidade de pessoas que devem embarcar neste ponto. Este valor raramente será exato: o jogo usa um valor aleatório para mais ou para menos dependendo das configurações do jogador, aquela porcentagem de passageiros nas opções. Para usar um número mais alto é necessário ter “vagas” suficientes no ponto (aqueles quadradinhos .000, .001 etc no Blender), caso contrário as pessoas podem aparecer sobrepostas ou ocorrer algum bug inesperado.

**DICA: EM PONTOS DE LINHAS LOTADAS, TERMINAIS, SHOPPINGS, PARQUES ETC COLOQUE MAIS PASSAGEIROS! JÁ EM ÁREAS MENOS POPULOSAS E RUAS MORTAS, DEIXE MENOS.**

A seção **[from\_3d]** define algumas configurações que serão lidas em conjunto dos modelos 3D definidos lá no Blender.

**readFrom3D=1** deve sempre ser 1 nesse caso. Nos protótipos internos seria 0 porque as coordenadas seriam definidas manualmente no txt, mas ler a partir do 3D é infinitamente mais prático para fazer o mapa. Não precisaremos ficar copiando e colando centenas de números, somente as rotações mesmo.

**prefix=nomedoponto** deve conter o nome do ponto, aquele nome interno que é usado no 3D: o mesmo nome antes do \_trigger ou antes do .000, .001 etc.

**IMPORTANTE: O VALOR DO ITEM PREFIX DEVE COINCIDIR EXATAMENTE COM O NOME USADO NO 3D! SE FOR DIFERENTE O PONTO PODERÁ SER IGNORADO OU BUGAR ALGUMA OUTRA COISA INESPERADA. PARA EVITAR ERROS NESSA HORA OS NOMES INTERNOS DOS PONTOS NÃO PODEM TER ESPAÇOS, CARACTERES ESPECIAIS, ACENTOS ETC. USE O NOME AMIGÁVEL PARA OS JOGADORES NA PROPRIEDADE “NAME” LÁ EM CIMA, CUIDANDO DESTE INTERNO PARA USO DO SISTEMA APENAS.**

**maxPathsToCheck=30** define o tanto de posições de passageiros a checar para este ponto. Caso seu ponto tenha mais de 30 posições de passageiros, aumente este valor aqui, para evitar que as demais sejam ignoradas. Este é um item de otimização. O jogo não tem como adivinhar quantas posições têm, ele precisa ler todas, a partir do 0, e ir somando 1... Até não achar mais.

Esse item permite que a contagem pare no número definido, evitando sobrecarregar o processamento do jogo no carregamento do mapa, especialmente quando tiver muitos pontos.

**defPaxRotY=37** define a rotação padrão dos passageiros. É a rotação no eixo vertical, indicando para onde estarão apontando. Normalmente no Blender será a rotação em Z, lembrando que na engine que usamos é em Y. Isso indica para onde a pessoa vai estar apontando, então o valor irá variar de rua para rua. Recomendamos testar com valores como 0, 90, -90 ou 180, e a partir deles fazer algum ajuste mais fino.

Por fim temos as seções opcionais, que são basicamente definidas pelo número do path entre colchetes. Elas têm apenas o parâmetro **rotY**, permitindo definir uma rotação individual para cada pessoa. No exemplo:

[006]

rotY=180

[005]

rotY=90

Isso indica que o passageiro na posição 006 estará rotacionado a 180 graus enquanto que o da 005 estará a 90. Todos os demais não definidos aqui usarão aquela **defPaxRotY** definida no [from\_3d]. Normalmente você não precisa se preocupar em definir isso, quase sempre a **defPaxRotY** já basta. Pode apagar ou nem precisa colocar esses números dos paths individuais.

**DICA: APAGUE AS ROTAÇÕES INDIVIDUAIS CASO VOCÊ COPIE O TXT DO MAPA DE EXEMPLO!**  
**ESSES ITENS COM O NÚMERO ENTRE COLCHETES DEVEM SER CONFIGURADOS CASO A CASO. SE NÃO FOR UTILIZÁ-LOS, REMOVA-OS DO ARQUIVO. O MESMO VALERÁ PARA AS PROPRIEDADES DOS CAMINHOS DE CARROS, ÔNIBUS OU TRENS DEPOIS. SÃO APENAS EXEMPLOS POR AGORA.**

## Configurando os pedestres

Vamos primeiro para a parte teórica, fundamental na compreensão de como o sistema funciona. As marcações das áreas por onde andam os pedestres, carros e trens têm uma configuração bastante parecida. Irão mudar os nomes das pastas e alguns parâmetros entre eles, mas a ideia geral é a mesma para os três.

Eles andam em uma rota predefinida que funciona de forma linear, num path (caminho) formado por vários pontos. Basicamente os pontos são ligados em sequência. Os caminhos funcionam sempre em uma direção: do ponto anterior para o próximo.

Seria mais ou menos como um trecho positivo de um eixo do plano cartesiano:



Só que no caso podendo fazer curvas, pegar subidas ou descidas, etc. Mas sempre em sequência. Sabendo isso, você já sabe mais ou menos como fará os caminhos no jogo: posicionando objetos com determinado nome e uma sequência!

É um processo manual um pouco demorado, apesar de não ser difícil no conceito. Quem sabe futuramente teremos alguma ferramenta para automatizar a produção dos paths, sabemos que atualmente é bem chato. A mesma dificuldade nós passamos no passado nos mapas nativos. Não desanime, é muito legal ver o mapa com o tráfego fluindo! Geralmente o resultado da dedicação vale muito a pena.

Alguns pontos são também “spawners”, que fazem “nascer” um objeto andante ali que já sai se movendo em direção ao próximo ponto. Ao chegar no final do caminho eles têm duas opções: sumir, desaparecer... Ou assumir um novo caminho à frente, caso achem algum que se inicie exatamente onde o anterior termina.

Para fins de otimização o jogo só mostra os andantes (pessoas, veículos ou trens) que estiverem num certo raio de distância do jogador. Seria um desperdício de recursos ficar renderizando um objeto pequeno a 5km de distância. Por outro lado, é muito ruim dar spawn (fazer o andante aparecer) bem na cara do jogador. Eles precisam aparecer naturalmente, vindo de algum lugar. Por isso eles costumam não aparecer logo de cara na frente do jogador.

Há um nível de aleatoriedade nos spawners para que não apareça sempre no mesmo instante ou muito sobrepostos. Você pode controlar a frequência da aparição para criar áreas mais ou menos densas. É possível também reduzir a quantidade de spawners, permitindo que apareçam menos objetos ao longo do tempo. Isso será útil para os trens, já que normalmente vai passar um depois de alguns minutos, não um atrás do outro como os pedestres nas grandes cidades.

**NOTA: OS PEDESTRES E OS TRENS NO PROTON NÃO IDENTIFICAM COLISÃO COM OS OBJETOS À FREnte POR DESIGN MESMO, DIFERENTE DOS VEÍCULOS COMO CARROS E ÔNIBUS. ISSO FOI FEITO PARA FINS DE OTIMIZAÇÃO, O QUE PERMITE RODAR COM BASTANTE PEDESTRES. O SISTEMA ATUAL NÃO É PROJETADO PARA TRAVESSIAS NA PISTA, CASO CONTRÁRIO PODERIA LOTAR DE PESSOAS PASSANDO QUE NUNCA DARIAM PASSAGEM PARA O ÔNIBUS (COMO PODE SER OBSERVADO NO METRÔ CARRÃOZINHO, NO MAPA ARICANDUVA). TENTE EVITAR ESTAS PASSAGENS.**

Basicamente cada circuito, path ou caminho (que são a mesma coisa) tem sua configuração independente dos demais. Cada um deve ter, portanto, um nome único. Vale a mesma dica para os pontos de ônibus: evite usar nomes comuns que possam ser confundidos com alguns outros objetos do mapa. É legal usar um prefixo e algum nome curto para cada path.

Os paths dos pedestres serão configurados com objetos no modelo 3D para pegar as posições, começando no .000. Por exemplo:

xxCalcada1.000, xxCalcada1.001, xxCalcada1.002 etc

ruaOutra.000, ruaOutra.001, ruaOutra.002 etc

Evite usar nomes que possam existir em outros objetos, como cube, door, street, etc. Algum prefixo exclusivo no início deve ajudar a evitar isso. Não queremos forçar o uso de um prefixo

só, então fica a seu critério escolher e gerenciar isso nos seus mapas. Se por acaso o nome de algum objeto coincidir com o nome de um caminho poderá dar ruim, já que o jogo não irá saber qual é qual. Por exemplo, se tiver uma peça de rua que chama rua1.003, resultado de uma duplicação de algum objeto... E um caminho configurado como rua1.000, rua1.001, rua1.002... Na hora de montar o caminho o jogo poderia ler a peça de rua como sendo parte do caminho, gerando um efeito desastroso: o pedestre, carro ou trem seria movido para outro lugar nada a ver e a peça de rua sumiria.

Assim como nos posicionadores do ponto de ônibus, prefira usar planos simples sem materiais, quando possível, para deixar mais leve. Mas se o mapa for pequeno não tem problema usar cubinhos não. Se vários paths se cruzarem, pode ser interessante criar um material sem textura, só com uma cor, para que você possa ver no Blender de forma mais prática. Os modelos 3D e os materiais desses caminhos serão destruídos no carregamento. Por isso quanto mais leves forem, melhor, menos desperdício lendo os dados deles.

**DICA: O NOME TÉCNICO DOS PONTOS DOS CAMINHOS É “WAYPOINT”. É COMUM TRATAR ESTES PONTOS COMO WAYPOINTS, LITERALMENTE “PONTOS DE CAMINHO” SE FÓSSEMOS TRADUZIR LITERALMENTE.**

Quanto aos spawners, uma observação importante: deixar todos os waypoints como sendo spawners é uma boa medida para testar mais rapidamente o mapa ou preencher áreas densas, como praças importantes, terminais movimentados etc. Porém isso pode ter um impacto negativo no peso do jogo, incluindo tanto memória como processamento.

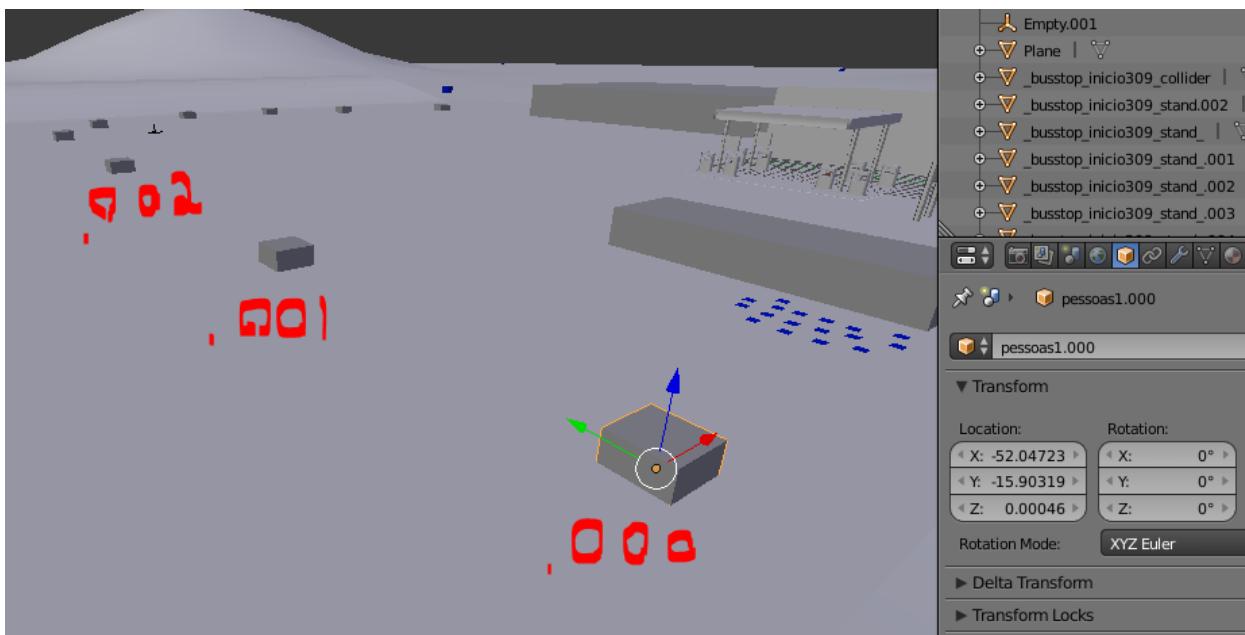
Os spawners precisam fazer vários cálculos a cada temporada (o tempo é definido no intervalo de spawn deles). Esses cálculos são leves, mas pode ficar complicado caso sejam milhares ao mesmo tempo. Em mapas muito grandes ou com muitas vias longas de muitas faixas, se todos os paths forem spawners, o desempenho do jogo pode começar a cair de forma significativa. Por limitações inerentes à forma como a engine funciona, todos os cálculos feitos no sistema rodam no thread principal da aplicação, disputando processamento de um único núcleo do processador. Quanto mais operações simultâneas ele precisar fazer, mais pesado pode ficar, demorando mais para renderizar o próximo frame.

Normalmente não há com o que se preocupar, mas vale ficar atento. Se o mapa tiver muitos quilômetros ou muitas ruas isso pode começar a ser um problema. Na nossa experiência do Proton deu para ver que na maioria das situações não será um problema. O antigo mapa do corredor da linha 4520 do Aricanduva era feito numa cena só, exatamente como os mods, tinha três faixas em cada sentido... E o desempenho não era comprometido por isso. Mas se for um mapa com dez vezes aquele tamanho a situação poderia sair do controle. Então fique atento, quanto menos spawners, em geral, melhor.

Em curvas é necessário definir mais pontos para que os objetos façam uma curva suave. Em áreas retas dá para dar um espaçamento maior entre os pontos. A movimentação dos veículos ainda passará por vários aprimoramentos ao longo da vida útil do projeto, mas os caminhos provavelmente não precisarão mudar.

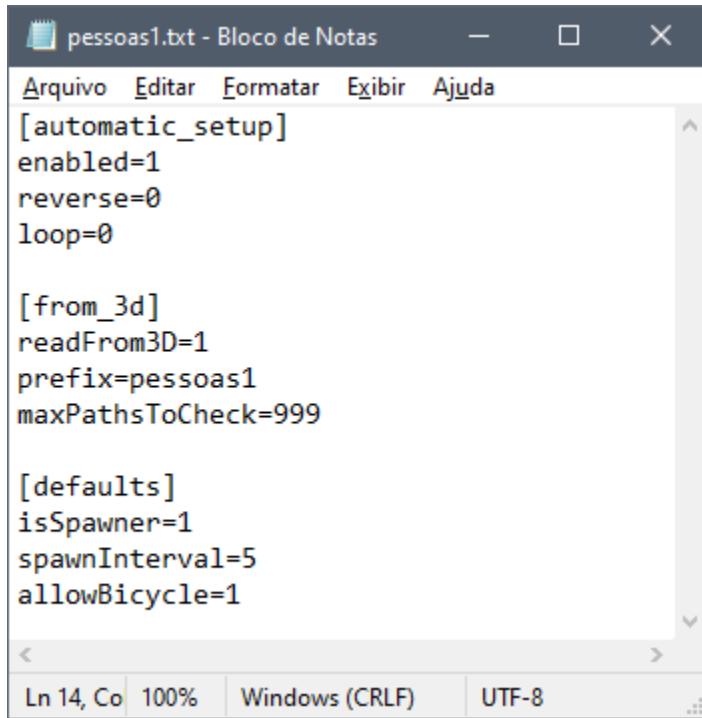
## Agora vamos à prática!

Basicamente não tem segredo na hora de colocar no Blender: crie um plano ou um cubo, reduza a escala dele para não atrapalhar outras coisas, e nomeie com um nome único seguido de .000. Por exemplo, xxCalcada1.000 (manteremos o xx para não misturar com o zz dos pontos, mas isso é só uma sugestão ou exemplo). A seguir, duplique este objeto (CTRL + D no Blender) e posicione o novo um pouco mais à frente. E vá fazendo isso... Ele vai preenchendo automaticamente os números seguintes para a gente. Por isso usamos essa coisa de terminar em .000, facilita demais! Pelo menos no Blender 2.79, tomara que seja sempre assim.



Feito isso na área desejada, vamos para o txt! Crie um txt novo na pasta **aippeople**, dentro da pasta dos modelos (a **modelsDir** que é definida no txt principal do mapa). O nome desse txt é indiferente, mas é bom que seja o mesmo do caminho, para facilitar a organização futura caso você precise fazer alterações depois.

Esse arquivo tem a seguinte estrutura:



```
[automatic_setup]
enabled=1
reverse=0
loop=0

[from_3d]
readFrom3D=1
prefix=pessoas1
maxPathsToCheck=999

[defaults]
isSpawner=1
spawnInterval=5
allowBicycle=1
```

A primeira seção **[automatic\_setup]** tem estes itens:

**enabled=1** é muito importante, deixe sempre assim. O sistema fará uma configuração automatizada ligando um ponto ao outro, até o último. Isso facilita imensamente as coisas, caso contrário teríamos que indicar qual é o próximo caminho manualmente.

O parâmetro **reverse=0** é opcional. Deixar em 1 fará o caminho ser configurado no sentido oposto, do último ao primeiro.

**OBSERVAÇÃO: PARECE EXISTIR UM BUG ONDE ESTA CONFIGURAÇÃO É IGNORADA. NÃO RECOMENDAMOS DEIXAR REVERSE EM 1 POR AGORA, TODAVIA É PARA ISSO QUE ELA FOI CRIADA.**

Por fim, **loop** define se o caminho será feito em loop. Deixando em 0 (false, o padrão) significa não, ou seja: ao acabar o caminho ele de fato acaba. A pessoa (ou carro, trem etc) irá sumir ao

chegar no último waypoint. Deixando loop=1 então o último ponto será ligado ao primeiro. Isso é muito útil em terminais e praças, ou algum quarteirão todo fechado de pessoas, mas não faz muito sentido nas ruas normais, que geralmente são lineares. Se loop tiver o valor 1, as pessoas ficarão andando em círculos no caminho todo.

**CUIDADO: USAR O LOOP EM CAMINHOS PEQUENOS NÃO É RECOMENDÁVEL! OS SPAWNERS CONTINUARÃO ATIVOS, PODERÃO LOTAR DE PESSOAS CADA VEZ MAIS, JÁ QUE ELAS NÃO SOMEM AO CHEGAR NO FINAL DO LOOP... DEVIDO ESTAREM POTENCIALMENTE PRÓXIMAS AO JOGADOR... O LOOP SERÁ BOM PARA ÁREAS UM POUCO MAIORES, ONDE A PRÓPRIA DISTÂNCIA FAZ AS PESSOAS SUMIREM, CASO DO TERMINAL CARRÃOZINHO NO MAPA ARICANDUVA, POR EXEMPLO. ISSO NÃO É ADEQUADO PARA TERMINAIS PEQUENOS.**

A seção **[from\_3d]** é muito parecida com a que vimos nos pontos de ônibus:

**readFrom3D=1** será sempre 1, para ler os caminhos a partir do modelo 3D.

**prefix** define o nome dos objetos do caminho, sem o ponto e os números. Nos exemplos citados e/ou nos prints, pessoas1, xxCalcada1 etc. O nome do txt é indiferente, mas este prefixo deve coincidir exatamente com o nome utilizado nos objetos no modelo 3D.

**maxPathsToCheck=999** é parecido com o do ponto também, porém com limite padrão em 999 em vez de 30 aqui. Normalmente você não precisará alterá-lo, a menos que o caminho que estiver configurando tenha mais de 999 pontos. Caso contrário os demais seriam ignorados.

A configuração dos waypoints tem uma nova seção muito importante: **[defaults]**. Esta definirá as opções padrões válidas para todos os pontos do caminho. Isso é feito para que você não precise configurar todos eles, já que os caminhos geralmente terão centenas ou milhares de pontos no mapa todo.

**isSpawner** define se o caminho é ou não um spawner. Se tiver o valor 1, sim, serão spawners, podendo fazer aparecer pessoas, carros ou trens ali. Se tiver o valor 0 eles não serão spawners, aí cada path poderá ser configurado como um spawner de forma independente depois. Normalmente para pedestres e veículos usamos isSpawner=1, mas para trens, 0, para não ficarem aparecendo trens sobrepostos.

**spawnInterval** define a base do intervalo com o qual o código do spawner irá rodar, em segundos. Deixando em 5, a cada 5 segundos (com alguma variação aleatória leve) os paths desse caminho tentarão dar spawn no objeto andante. Não é recomendável deixar um valor muito baixo pois pode trazer problemas de performance. Aumentar o tempo faz com que demore mais para aparecer as pessoas, permitindo configurar áreas menos densas, como de ruas secundárias ou bairros mais sossegados.

**allowBicycle** define se pode ou não aparecer ciclistas nesse path. Deixe com 1 para permitir, ou 0 para negar. Normalmente paths de terminais terão o valor 0, para evitar que apareçam ciclistas andando nas plataformas. Todavia, se algum outro path fizer ligação com o terminal e ele permitir ciclistas, os ciclistas poderão terminar de passar por dentro do terminal normalmente.

**OBSERVAÇÃO:** PELAS REGRAS DE TRÂNSITO OS CICLISTAS DEVEM ANDAR NAS RUAS, TODAVIA NO PROTON ESTÃO PROGRAMADOS PARA APARECEREM JUNTOS AOS PEDESTRES, O QUE PODE FAZER ELES APARECEREM EM CALÇADAS, O QUE NÃO É MUITO DIFERENTE DA NOSSA REALIDADE... NO MOMENTO NÃO É POSSÍVEL CONFIGURAR CICLOVIAS EXCLUSIVAS.

A seção **[defaults]** define o comportamento para todos os waypoints do caminho. Se você quiser fazer uma parte diferenciada em algum trecho, por exemplo, reduzindo a densidade, poderá alterar as propriedades de um caminho em específico. No final do arquivo crie uma entrada entre colchetes com o número do path, e a seguir, as propriedades dele. Por exemplo, supomos que no path intermediário de número 512 você quer que demore mais para aparecer pessoas. Então colocaria:

```
[512]  
isSpawner=1  
spawnInterval=50  
allowBicycle=0
```

Repita isso para outros paths que possam interessar. Isso vai ser muito útil no caso dos trens, onde geralmente o isSpawner do **[defaults]** será 0, e apenas um path ou outro do caminho terá o isSpawner=1.

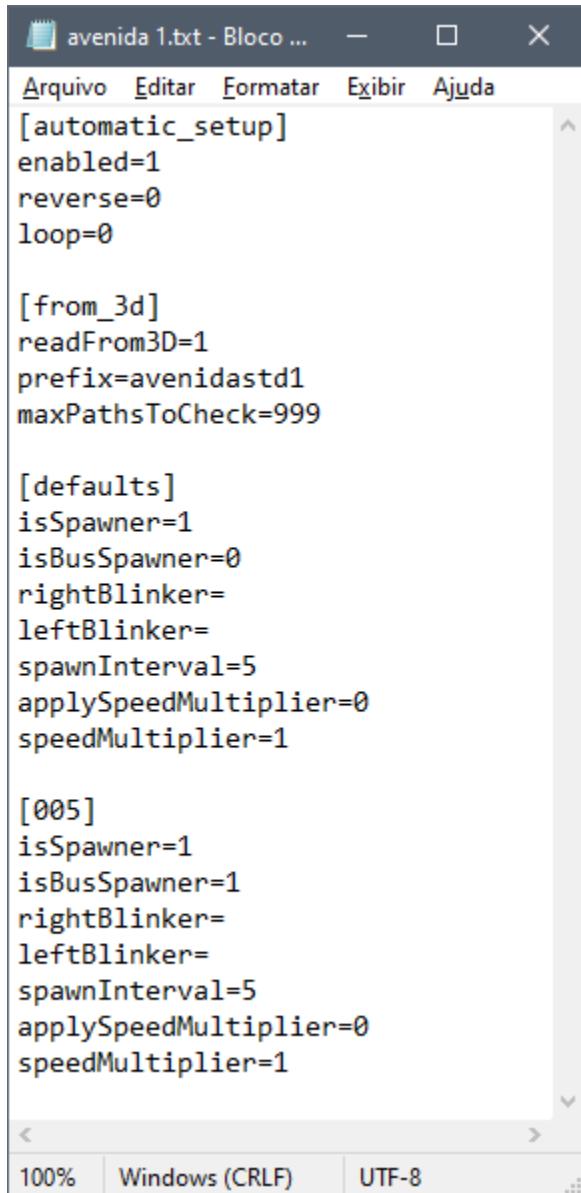
Para criar mais caminhos não interligados, é só fazer o mesmo procedimento, escolhendo outro nome para eles!

**IMPORTANTE: CASO O ÚLTIMO WAYPOINT DE UM CAMINHO ESTEJA PRATICAMENTE SOBREPOSTO AO PRIMEIRO WAYPOINT DE OUTRO CAMINHO, O SISTEMA ENTENDERÁ ISSO COMO UMA CONTINUAÇÃO! ISSO É ÚTIL AO LIGAR ÁREAS DIFERENTES SEM PRECISAR FAZER CONFIGURAÇÕES COM NÚMEROS MUITO ALTOS.**

**MUITO IMPORTANTE: NÃO POSICIONE OS WAYPOINTS MUITO PRÓXIMOS! SE ELES FICAREM MUITO COLADOS UNS NOS OUTROS, OS OBJETOS SE MOVENDO PODERÃO FICAR PARADOS OU NÃO VÃO CONSEGUIR GIRAR DIREITO. NAS CURVAS E BORDAS É NECESSÁRIO USAR MAIS PONTOS PARA DEIXÁ-LAS SUAVES, MAS NÃO EXAGERE. MUITOS WAYPOINTS PRÓXIMOS NO MESMO CAMINHO PODEM BUGAR O SISTEMA.**

## Configurando os veículos do tráfego

Depois que você aprendeu a colocar pedestres, os carros serão moleza! É exatamente a mesma coisa, com algumas propriedades diferenciadas. A pasta dos txt para os veículos é a **aivehicles**, que também fica dentro da pasta dos modelos.



The image shows a screenshot of a Windows Notepad window titled "avenida 1.txt - Bloco de Notas". The window contains configuration code for traffic vehicles. The code is organized into sections: [automatic\_setup], [from\_3d], [defaults], and [005]. Each section contains various parameters with their values. The code is as follows:

```
[automatic_setup]
enabled=1
reverse=0
loop=0

[from_3d]
readFrom3D=1
prefix=avenidastd1
maxPathsToCheck=999

[defaults]
isSpawner=1
isBusSpawner=0
rightBlinker=
leftBlinker=
spawnInterval=5
applySpeedMultiplier=0
speedMultiplier=1

[005]
isSpawner=1
isBusSpawner=1
rightBlinker=
leftBlinker=
spawnInterval=5
applySpeedMultiplier=0
speedMultiplier=1
```

At the bottom of the Notepad window, there are status indicators: "100%", "Windows (CRLF)", and "UTF-8".

IMPORTANTE: LEMBRE-SE DE USAR NOMES ÚNICOS! NÃO MISTURE OS PATHS DE PESSOAS COM CARROS OU TRENS, PARA EVITAR SE PERDER OU VER UM OBJETO APARECENDO ONDE NÃO DEVERIA. A EQUIPE DO JOGO NÃO TERÁ TEMPO PARA FICAR REVISANDO SEUS MAPAS TENTANDO CORRIGIR ERROS CAUSADOS POR MÁ CONFIGURAÇÃO, ENTÃO POR FAVOR, SEJA PACIENTE AO ORGANIZAR A SUA ESTRUTURA. ANTES DE RELATAR UM POTENCIAL BUG DO JOGO, REVISE TUDO PARA TER CERTEZA QUE NÃO É PROBLEMA NO SEU MAPA PRIMEIRO.

Para os carros, as propriedades das seções **[automatic\_setup]** e **[from\_3d]** são exatamente as mesmas das pessoas. As mudanças ficam nas propriedades individuais inerentes ao sistema deles. Elas podem ser definidas na **[defaults]**, para valer para todos os waypoints deste caminho; bem como com **[número]**, valendo só para um waypoint específico.

**isSpawner** é a mesma coisa das pessoas: 0 desativado e 1 ativado, para definir se neste caminho devem aparecer os veículos ou não.

**isBusSpawner** é um pouco diferente: é um spawner secundário que só serve para os ônibus. Se você quer que apareça bastante ônibus, pode deixar **isBusSpawner=1** no **[defaults]**. Normalmente isto não é recomendável para todos os lugares, já que não é tão normal lotar de ônibus em todos os cantos de todas as ruas, a menos que o **spawnInterval** seja mais alto. Em alguns casos recomendamos deixar isso em 0 e definir alguns pontos aleatórios depois para aparecer os ônibus, colocando o número deles entre os colchetes e **isBusSpawner=1**.

**DICA: EM CORREDORES EXCLUSIVOS DE ÔNIBUS, DEIXE **isSpawner=0** NO **DEFAULTS** E **isBusSpawner=1**, ASSIM SÓ APARECERÃO ÔNIBUS ALI!**

**rightBlinker** define se a seta direita do veículo deve piscar enquanto ele estiver com este waypoint como alvo. Normalmente no **defaults** será sempre 0, podendo ser 1 para os paths das esquinas específicas, pegando pelo número deles entre colchetes.

**leftBlinker** é similar, porém para a seta esquerda. Deixe também em 0 no **defaults**, use apenas nos waypoints específicos.

**spawnInterval** define o intervalo do spawner, em segundos. Aumente este valor para dar um tempo maior entre um carro e outro que nasceria ali, permitindo assim reduzir a densidade. É bom aumentar o intervalo de spawn dos paths de ruas secundárias ou mais mortas, ou em áreas onde o jogador precise entrar numa conversão apertada.

**applySpeedMultiplier** permite decidir se aplica ou não o multiplicador de velocidade, definido logo a seguir. O padrão é não, fica em 0. Se você quiser alterar a velocidade do caminho, deixando mais rápido ou mais lento, deixe em 1, e a seguir configure o multiplicador no próximo item **speedMultiplier**.

**speedMultiplier** define um multiplicador para a velocidade base dos veículos neste caminho. Ele pode valer tanto no defaults, que serve para todos, como em paths individuais, permitindo que os veículos andem mais devagar numa região de escola ou subida, por exemplo.

A velocidade dos carros no Proton ainda é algo a ser melhor trabalhado. Todavia este multiplicador permite alterar a velocidade base do path em questão. Não use zero, utilize algum valor quebrado entre 0 e 2. Não é legal exagerar. Se deixar muito rápido, os carros com o sistema atual podem não conseguir frear a tempo em caso de algum obstáculo ou outro problema qualquer.

Colocar o **speedMultiplier** em 0.5 fará o limite de velocidade ser cerca de metade do atual da via, e em 2 fará ser o dobro. Note que cada carro tem um fator de personalização interno aleatório, de forma que simule um pouco da realidade: nem todos eles têm a mesma arrancada e velocidade alvo igual. Isso pode ser observado em alguns cruzamentos do mapa Aricanduva, como na área do Metrô Carrãozinho: ao abrir o semáforo eles saem de maneira não uniforme, com alguns acelerando mais do que outros. Isto não é definitivo e ainda passará por muitos aprimoramentos, mas já é bastante interessante.

Lembre-se que para o multiplicador de velocidade fazer efeito, o **applySpeedMultiplier** deve ter o valor 1, para ser ativado.

**DICA: DEIXE UMA VELOCIDADE UM POUCO MAIS RÁPIDA EM FAIXAS EXPRESSAS DE RODOVIAS, POR EXEMPLO, E MAIS LENTA NAS FAIXAS LOCAIS!**

## Configurando as pinturas dos ônibus

Na fase 2 dos mods de mapas não é possível modificar os veículos do tráfego. Isso vale para os ônibus também. Este recurso é desejado porém não confirmado, dada a complexidade para fazer e o potencial peso que trará ao jogo.

Todavia, as skins do PBC no tráfego são personalizáveis!

Você pode colocar as skins dentro da pasta **skins/0/pbc**. A do visstaLO ainda não é confirmada, talvez ele será removido para liberar mais RAM para o sistema de mapas.

Basta colocar as imagens desejadas dentro da pasta skins/0/pbc que o jogo deverá sorteá-las aleatoriamente quando um pbc for aparecer.

**DICA: USE UMA SKIN SÓ, OU ENTÃO, ALGUMAS POCAS! LOTAR DE SKINS FARÁ O JOGO CONSUMIR MAIS RAM, NORMALMENTE AS SKINS SÃO IMAGENS GRANDES. OS USUÁRIOS PODEM TROCAR AS SKINS FACILMENTE SUBSTITUINDO AS IMAGENS DA PASTA.**

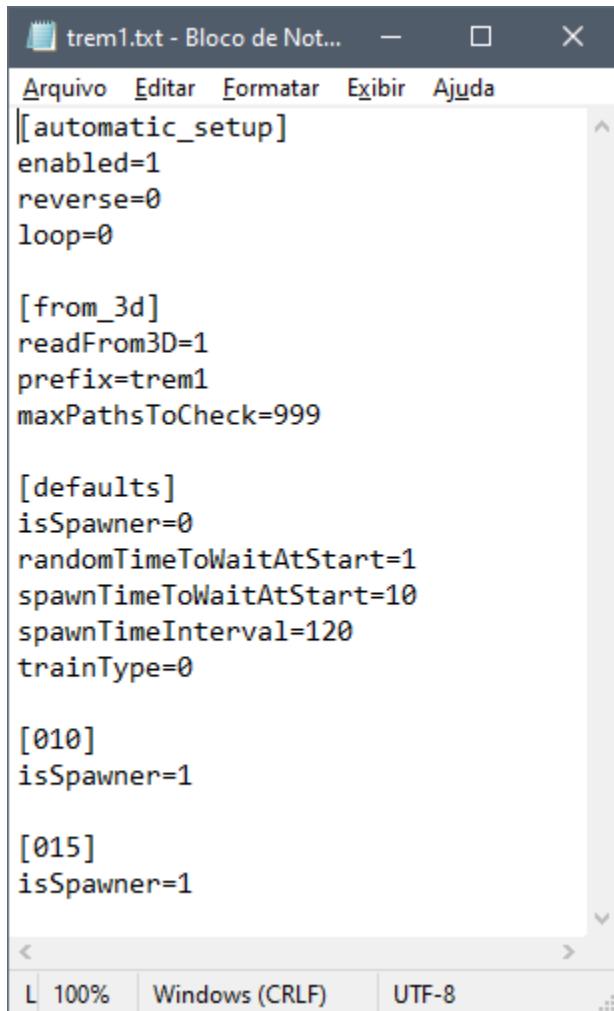
As skins devem ser imagens PNG, de preferência, mantendo aquela observação de não usar acentos nem caracteres especiais no nome.

A base de skin do pbc está aqui:

<http://omsi.viamep.com/proton/base-skins-pbc-protonbussimulator.zip>

## Configurando os trens

A configuração dos trens é quase idêntica às dos pedestres e carros, com a diferença na recomendação: é bom deixar o isSpawner do [defaults] sempre em 0, e definir apenas alguns pontos bem espaçados como spawners manualmente. Os txts dos paths de trens ficarão na pasta **aitrains**, dentro da pasta dos modelos.



```
trem1.txt - Bloco de Notas - Arquivo Editar Formatar Exibir Ajuda

[[automatic_setup]]
enabled=1
reverse=0
loop=0

[from_3d]
readFrom3D=1
prefix=trem1
maxPathsToCheck=999

[defaults]
isSpawner=0
randomTimeToWaitAtStart=1
spawnTimeToWaitAtStart=10
spawnTimeInterval=120
trainType=0

[010]
isSpawner=1

[015]
isSpawner=1

L 100% Windows (CRLF) UTF-8
```

De parâmetros exclusivos dos trens, temos estes:

**randomTimeToWaitAtStart** deve ter o valor 0 ou 1 só, é um valor booleano (true/false, verdadeiro ou falso). Se for verdadeiro (1), o sistema irá esperar um valor aleatório entre 0 e o intervalo de spawn definido para aparecer o primeiro trem assim que o mapa for carregado. Por exemplo, supondo que o intervalo seja de 120 segundos (dois minutos)... Se **randomTimeToWaitAtStart** for igual a 1 então o primeiro trem poderá aparecer a qualquer momento desde que o mapa foi carregado, de alguns poucos segundos, ou um minuto, ou quase dois... Será aleatório nesse intervalo. Normalmente isso é útil para que o trem não apareça sempre ao mesmo tempo caso o jogador dê spawn próximo a uma estação ou trilho com ponto de spawn. Assim dará uma sensação de aleatoriedade: algumas vezes o trem aparece logo que o jogador iniciou o mapa, mas em outras vezes demora mais... É bem útil.

**spawnTimeToWaitAtStart** será usado quando o **randomTimeToWaitAtStart** tiver o valor 0. Nesse caso o primeiro trem dará spawn no tempo definido, por exemplo, depois de 20 segundos que o jogo for carregado.

**spawnTimeInterval** define o intervalo regular do trem, em segundos (levemente aleatório), depois que o primeiro apareceu. Recomendamos algo a partir de 120 (dois minutos), que é um intervalo razoável para sistemas de trens ou metrôs modernos. Em alguns locais pode ser bem mais. Note que o tempo é definido em segundos, então multiplique ou divida por 60 caso você queira fazer a conta em minutos.

Estas configurações permitem um bom nível de aleatoriedade, já que nem sempre o usuário verá o trem passando exatamente no mesmo local. Note que para que o temporizador inicial faça mais sentido, o jogador deve ter iniciado o mapa num raio próximo ao spawner do trem. Ao dar spawn em outro ponto e se aproximar dirigindo, a aleatoriedade será mais alta, melhor ainda.

Por fim, **trainType** define o tipo de trem que aparecerá no lugar. O valor padrão é 0, recomendamos deixar em 0 até que alguma atualização futura traga outros tipos de trens. Este valor permitirá diferenciar trens de carga, suburbanos (tipo CPTM) e metrô.

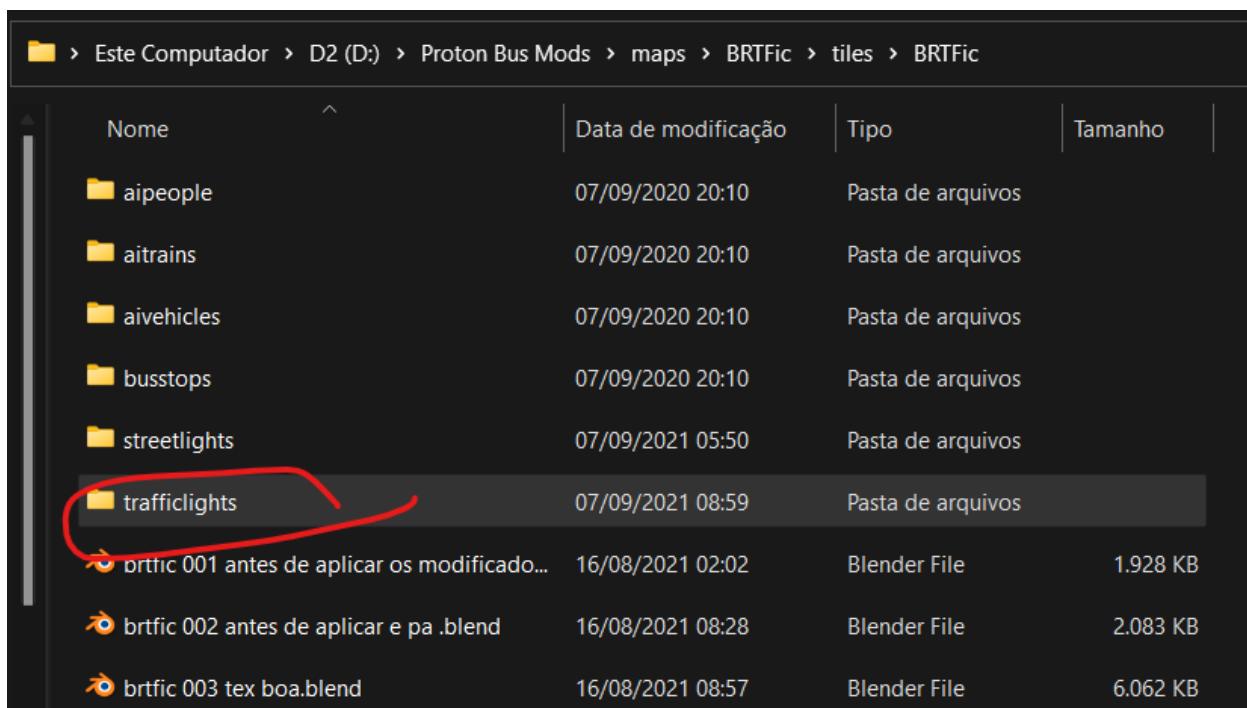
IMPORTANTE: OS TRENS NÃO TÊM DETECÇÃO DE FÍSICA COMO OS VEÍCULOS! ELES PASSARÃO POR CIMA DE QUALQUER COISA QUE ESTIVEREM NO CAMINHO. NÃO SÃO SUPORTADAS, PORTANTO, PASSAGENS EM NÍVEL.

TOME CUIDADO: POR FAVOR, POSICIONE OS SPAWNERS DE TRENS COM UMA BOA DISTÂNCIA ENTRE ELES! NORMALMENTE A CADA UM OU DOIS QUILÔMETROS, OU MAIS, NÃO MUITO PRÓXIMOS! SE VOCÊ POSICIONAR UM MONTE DE SPAWNERS DE TRENS MUITO PRÓXIMOS OU USAR `isSpawner=1` NO DEFAULTS, UM MONTE DE TRENS PODERÃO APARECER SOBREPOSTOS. O SISTEMA FOI PROJETADO PARA TER OS SPAWNERS DE TRENS BEM ESPAÇADOS PARA PODER FUNCIONAR BEM.

## Fase 3: Semáforos

Os semáforos envolvem uma configuração tanto nos txt como no modelo 3D. O txt definirá as propriedades da “máquina” do semáforo, com os detalhes de quais vias serão bloqueadas ou liberadas, e as fases das luzes. As peças no 3D indicarão as posições dos bloqueios (triggers) e das luzes visuais.

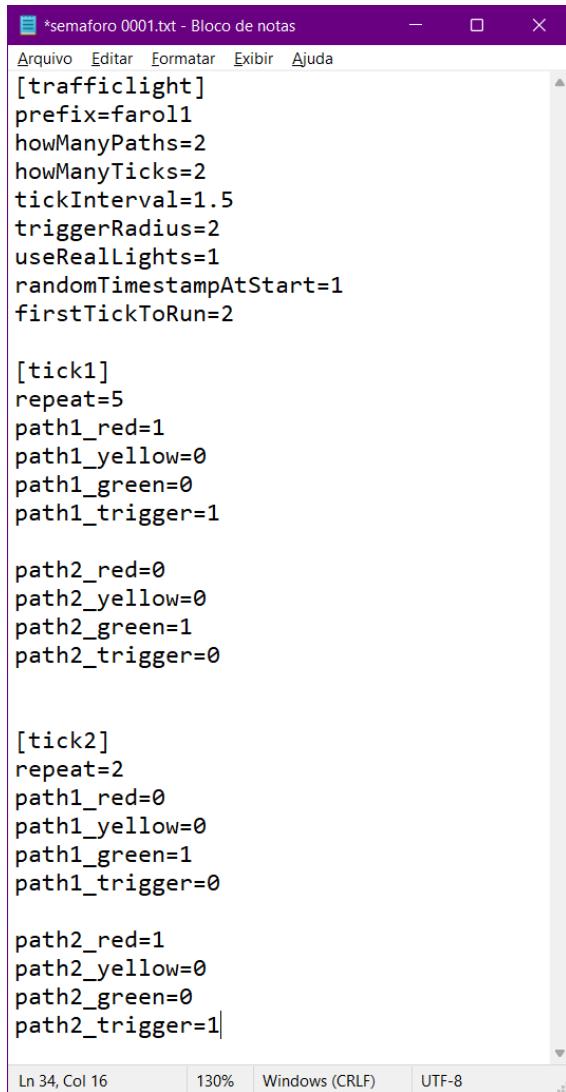
Os arquivos de texto de configuração dos semáforos devem ficar na pasta **trafficlights**, dentro da pasta do **tile** (onde ficam os modelos e as pastas aipeople, aivehicles, busstops etc).



Nome	Data de modificação	Tipo	Tamanho
aipeople	07/09/2020 20:10	Pasta de arquivos	
aitrains	07/09/2020 20:10	Pasta de arquivos	
aivehicles	07/09/2020 20:10	Pasta de arquivos	
busstops	07/09/2020 20:10	Pasta de arquivos	
streetlights	07/09/2021 05:50	Pasta de arquivos	
<b>trafficlights</b>	07/09/2021 08:59	Pasta de arquivos	
brtfic 001 antes de aplicar os modificado...	16/08/2021 02:02	Blender File	1.928 KB
brtfic 002 antes de aplicar e pa .blend	16/08/2021 08:28	Blender File	2.083 KB
brtfic 003 tex boa.blend	16/08/2021 08:57	Blender File	6.062 KB

Dentro dessa pasta **trafficlights** ficará um txt para cada “máquina” dos semáforos. Semáforos que usarem a mesma estrutura podem compartilhar a mesma máquina, não há a necessidade de criar um txt para cada cruzamento não. Aliás é bom que nem seja feito isso mesmo por conta da performance, o ideal é reaproveitar um mesmo txt para vários cruzamentos caso sejam similares.

A estrutura básica dos txt dos semáforos é assim:



```
*semáforo 0001.txt - Bloco de notas
Arquivo Editar Formatar Exibir Ajuda
[trafficlight]
prefix=farol1
howManyPaths=2
howManyTicks=2
tickInterval=1.5
triggerRadius=2
useRealLights=1
randomTimestampAtStart=1
firstTickToRun=2

[tick1]
repeat=5
path1_red=1
path1_yellow=0
path1_green=0
path1_trigger=1

path2_red=0
path2_yellow=0
path2_green=1
path2_trigger=0

[tick2]
repeat=2
path1_red=0
path1_yellow=0
path1_green=1
path1_trigger=0

path2_red=1
path2_yellow=0
path2_green=0
path2_trigger=1

Ln 34, Col 16 130% Windows (CRLF) UTF-8
```

A seção **[trafficlight]** contém as definições gerais da máquina, e as seções **[tick...]** determinam o estado das luzes e bloqueios em cada um dos instantes.

Os semáforos no Proton funcionam assim: cada “tick” é um estado, uma captura de um dado momento. Em um “tick” o semáforo fica aberto para uma via e fechado para outra. No tick seguinte isso pode se inverter. O que era verde fica vermelho e vice-versa. Bastam dois ticks

para um semáforo bem rudimentar onde as luzes alternam apenas entre dois estados.

Normalmente as configurações usarão pelo menos quatro ticks por incluir o amarelo.

**Imagine isso:** no tick 1 a via principal está **aberta** e a lateral **fechada**. No tick 2 ambas ficam fechadas e a principal fica com o farol **amarelo**. No tick 3 a via 1 **fecha** e a via 2 **abre**. No tick 4 a via 1 continua fechada e a via 2 fica com o sinal amarelo. Aí volta para o tick 1, onde a via 1 **abre** e a via 2 fica **fechada**.

Cada rua que se cruza ou “via” será representada pelo nome **path**. Um cruzamento em X terá dois paths na configuração mais comum. Um cruzamento em T com um semáforo separado para virar à esquerda pode ter três paths, ou quatro, caso seja necessário abrir uma via de cada vez para não travarem no meio se a rua for apertada. Um cruzamento largo em X onde todas as direções são possíveis normalmente terá pelo menos quatro paths: principal e secundário, esquerda 1 e esquerda 2 (para quem vem de cada rua). É o que ocorre no mapa Fiktivdorf do Proton Bus urbano, por exemplo.

O tempo é definido pela propriedade **tickInterval**. A recomendação é deixar esse valor em 1 ou 2. Isso significa que cada tick terá 1 unidade de tempo (geralmente 1 segundo) caso seja 1. Esse valor pode usar um número decimal (com o ponto) para valores quebrados, que ajudam nos testes. Deixando **0.2** por exemplo, as fases vão passar bem rápido. Isso é ideal para testar durante o desenvolvimento sem ficar esperando o tempo todo.

Por padrão, cada tick irá durar o valor definido no **tickInterval** no começo do txt. O parâmetro **repeat** em cada tick permite manter ele ativo por mais tempo. Geralmente isso vai ser usado nas vias que ficam mais tempo abertas; e evitado (ou com um valor baixo) quando tiver um sinal amarelo.

**DICA: EM AVENIDAS IMPORTANTES COM BOA QUANTIDADE DE CARROS É BOM DEIXAR O AMARELO POR MAIS TEMPO, COMO NA VIDA REAL. ISSO PERMITE UM MAIOR TEMPO PARA “ESCOAR” TODOS OS CARROS ANTES DE ABRIR PRO OUTRO LADO, EVITANDO AS TRAVADAS QUE OCORREM NA AV. RADIAL DO MAPA ARICANDUVA, POR EXEMPLO.**

## Configurações gerais do motor do semáforo

Os principais parâmetros na chave **[trafficlight]** são estes:

**prefix** é o nome interno da peça no modelo 3D. Precisa ser um nome único para este semáforo que não seja confundido com nenhum outro objeto do mapa para não dar nenhum bug. O jogo vai buscar por todos os objetos que tiverem esse item no nome. É recomendável usar um nome simples sem acentos nem ç, como sempre. Algo como *semaforo001*, *farolavprincipal1*, etc.

**howManyPaths** define quantos paths ou vias esse semáforo controla. A maioria dos semáforos em X terá apenas dois paths: o principal (main) e o lateral (side). Portanto este item seria 2.

**IMPORTANTE: AS VERSÕES ATUAIS DO PROTON NÃO SUPORTAM OS SEMÁFOROS PARA PEDESTRES. PARA FINS DE SIMPLIFICAÇÃO DO PROJETO OS PEDESTRES ANDANTES NÃO VERIFICAM POR COLISORES À SUA FREnte, ELES PASSAM POR CIMA DE QUALQUER COISA. ISSO PERMITE TER UM MONTE DE PEDESTRES SEM TANTA PERDA DE PERFORMANCE. TALVEZ ALGUMA VERSÃO FUTURA PODERÁ MUDAR ISSO, MAS NÃO É PREVISTO. TODAVIA VOCÊ PODE ADICIONAR MAIS PATHS PARA AS LUZES DE PEDESTRES CASO QUEIRA, BASTA CONFIGURAR APENAS AS LUZES E IGNORAR OS TRIGGERS DENTRO DO MODELO 3D.**

**howManyTicks** define quantos ticks ou estados o semáforo vai ter. Um semáforo em X simples com apenas dois paths normalmente vai ter pelo menos quatro ticks:

- 1 - **verde** no main, **vermelho** no side
- 2 - **amarelo** no main, **vermelho** no side
- 3 - **vermelho** no main, **verde** no side
- 4 - **vermelho** no main, **amarelo** no side

Conforme a complexidade do semáforo aumentar pode ficar bastante complicado de configurar isso, mas não é nada tão diferente da vida real :p

**tickInterval** define o tempo do tick base, geralmente um bom valor será 1 ou 2 (um ou dois segundos). Se deixar um valor menor, como 0.1 ou 0.2 o semáforo irá trocar de fase bem rápido. Aumentando, irá demorar mais. A duração de cada tick individual é definida no parâmetro **repeat** dentro da configuração deles (veja mais abaixo). Se o tick for de 2 segundos, um semáforo com o repeat em 3 deve durar cerca de 6 segundos, e por aí vai.

**triggerRadius** define a largura do raio do trigger, o objeto invisível que vai bloquear a pista. Geralmente um bom valor é 1 ou no máximo 2. Se ele for muito grande pode acabar interferindo em outras vias próximas. Se for muito pequeno os carros podem acabar não “enxergando” e passar no farol vermelho. Ele deve englobar o ponto de parada do veículo. Logo mais na parte da configuração do 3D ele será exibido com mais detalhes.

**useRealLights** define se os semáforos usarão uma luz real ou não. 0 deixa desativado, 1 ativado. A luz real depende da plataforma e das configurações do jogo. Nas primeiras versões funciona somente no PC no modo ultra ou com o ciclo dia/noite com luz real. Enquanto este documento foi produzido, este recurso não estava disponível no celular por questões de performance. A configuração da cor e intensidade da luz é definida em outros campos a serem comentados mais abaixo.

**randomTimestampAtStart** permite randomizar o tempo rodando dentro do primeiro “tick”. É útil para que os motores dos semáforos rodem com tempos levemente diferentes. Se todos rodarem ao mesmo tempo, o jogo pode dar uma travada maior na hora de ativar as luzes. Esse parâmetro normalmente tem o valor 1 para ativar ou 0 para desativar. É recomendável deixar sempre em 1, a menos que você queira que algum semáforo fique mais sincronizado com algum outro.

**NOTE QUE A SINCRONIA ENTRE SEMÁFOROS DIFERENTES NÃO É GARANTIDA NEM PROMETIDA. ELA PODE VARIAR DEPENDENDO DE OUTROS RECURSOS DA ENGINE DO JOGO. EM TESE, SEMÁFOROS COM ESTE ITEM DESATIVADO (VALOR 0) E MESMO TEMPO E QUANTIDADE DE TICKS QUE OUTROS FICARIAM EM SINCRONIA. PORÉM DIFERENÇAS INTERNAS DA ENGINE COM OS NÚMEROS FLOAT CONFORME O TEMPO PASSA PODE PREJUDICAR ISSO.**

**firstTickToRun** define qual será o primeiro tick a rodar no carregamento do mapa. É opcional, normalmente tanto faz começar do primeiro ou não.

## Configurações de cada tick

Em cada tick teremos as configurações das luzes para cada um dos paths. O valor numérico para as luzes ali é binário: 0 significa desligado e 1 ligado (aceso).

Cada tick terá sua própria chave, basta usar números incrementais. Isso depende da propriedade **howManyTicks** definida acima. Para evitar que o jogo fique procurando números que não existem, ajuste este número em cada caso.

Observe este exemplo com apenas dois ticks, onde em um o path1 está bloqueado e o 2 liberado, e no outro inverte:

```
[tick1]
repeat=5
path1_red=1
path1_yellow=0
path1_green=0
path1_trigger=1
path2_red=0
path2_yellow=0
path2_green=1
path2_trigger=0
```

```
[tick2]
repeat=5
path1_red=0
path1_yellow=0
path1_green=1
path1_trigger=0
path2_red=1
path2_yellow=0
path2_green=0
path2_trigger=1
```

Você pode copiar os txt do mapa de exemplo disponível no site do jogo. Naturalmente a equipe do jogo não pode configurar os semáforos para você, mas devem ser liberados alguns exemplos de configurações populares.

DICA: NA CONFIGURAÇÃO DOS ESTADOS DAS LUZES E DO TRIGGER EM CADA TICK, O VALOR 0 PARA DESATIVADO É O VALOR PADRÃO. ENTÃO NÃO PRECISA ESCREVER QUANDO FOR 0. VOCÊ PODE FOCAR APENAS NAS LUZES OU TRIGGERS LIGADOS COM O VALOR 1. OS EXEMPLOS INCLUEM TANTO COM 0 COMO COM 1 PARA QUE A EXPLICAÇÃO FIQUE COMPLETA. PORTANTO, AO CRIAR SEMÁFOROS COMPLEXOS MANUALMENTE VOCÊ PODE POUPAR UM BOM TEMPO SE PREOCUPANDO APENAS COM AS QUE FICAM ACESAS, JÁ QUE AS NÃO DEFINIDAS FICARÃO APAGADAS POR PADRÃO.

## Configurações de luzes reais

Caso você queira configurar as luzes reais para o PC no modo ultra, que iluminam o asfalto (ficam bonitas na noite e/ou na chuva), pode opcionalmente colocar estas seções no txt:

### **[green\_light]**

**colorR** = cor do vermelho em RGB (de 0 a 1, consulte a parte da tabela de cores no material de ajuda em anexo)

**colorG** = cor do verde em RGB, de 0 a 1

**colorB** = cor do azul em RGB, de 0 a 1

**colorA** = cor do canal alpha (geralmente vai ser 1)

**intensity** = intensidade ou “força” da luz, de 0 a 1 (1 = máxima)

**range** = área de abrangência da luz, como um raio. Um bom valor pode ser de 10 a 30. Os mapas nativos do Proton Bus urbano usavam o valor 10 neste item.

Coloque esta chave fora das outras seções do txt, pode ser no final do arquivo ou antes dos ticks. Se colar dentro dos ticks ela pode ser ignorada ou produzir erro. Duplique este trecho renomeando a chave para as demais luzes:

**[red\_light]** e **[yellow\_light]** usarão a mesma estrutura para as luzes vermelha e amarela, respectivamente.

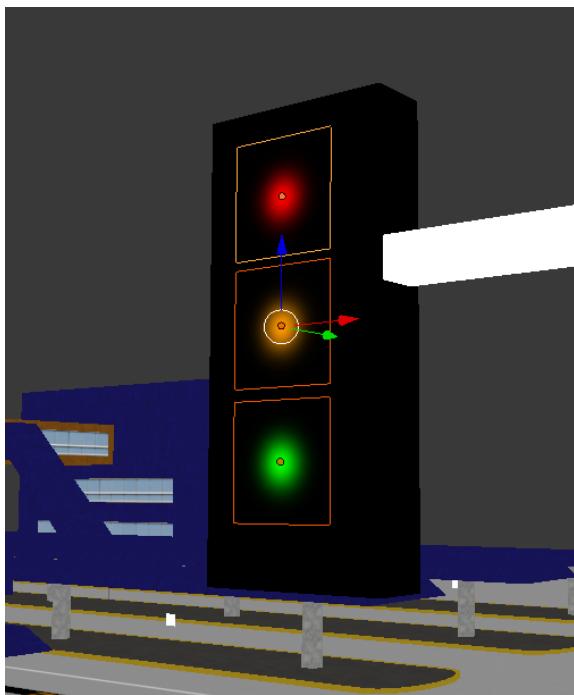
Caso você configure **useRealLights=1** na seção **[trafficlight]** e não defina estas propriedades individuais, não tem problema: o jogo vai usar valores padrões similares aos dos nativos para as luzes verde, vermelha e amarela.

## Colocando os semáforos no mapa no 3D

Definida toda a teoria, agora temos a parte mais fácil!

O mesh do semáforo pode ser um mesh normal, não tem segredo. A luz será um pequeno plano com uma textura. Recomendamos o shader additive em vez do emissive, ele parece ficar melhor para o semáforo.

Faça um mesh com o fundo escuro para a base do farol todo apagado. Os planos das luzes “soltos” ficarão com uma textura de fundo preto. O fundo será excluído nesse shader “additive”.



Nesse exemplo veja que as luzes são apenas “planos” com uma textura de fundo preto. Caso prefira usar o emissive então não use o fundo preto, use uma textura normal.

**IMPORTANTE: DEIXE UMA CERTA DISTÂNCIA ENTRE O PLANO E O FUNDO. SE FICAREM MUITO PRÓXIMOS OU COLADOS ELES PODEM FICAR PISCANDO. É O TRADICIONAL PROBLEMA Z-FIGHT MUITO COMUM NOS GAMES. QUANTO MAIS DISTANTE DA ORIGEM, PIOR AINDA.**

Os nomes dos planos das luzes são bem diretos:

\_farol1\_path1\_red\_additive\_ farol 1, path 1, luz vermelha

\_farol1\_path1\_green\_additive\_ farol 1, path 1, luz verde

\_farol1\_path1\_yellow\_additive\_ farol 1, path 1, luz amarela

\_farol1\_path2\_red\_additive\_ farol 1, path 2, luz vermelha... e por aí vai

O nome deve ser formado desta forma, de preferência em ordem:

**nome do farol definido no prefix lá no txt** **\_pathX** **cor** **\_additive**

Naturalmente você não vai ter apenas uma luz, mas várias! É só usar o mesmo esquema para as outras. Ao duplicar o objeto o Blender adiciona os números .002, .003 etc ao final do nome. Não tem problema, pode deixar eles. O que importa é ter as palavras chaves na ordem definida acima.

**SUPER DICA: REUTILIZAR SEMÁFOROS NO PROTON É MUITO FÁCIL! CASO ELES COMPARTILHEM O MESMO “MOTOR”, BASTA DUPLICAR AS LUZES E O TRIGGER E COLOCAR NOS DEMAIS LUGARES. ELES VÃO FUNCIONAR EM CONJUNTO COMPARTILHANDO O MOTOR.**

Qual via será o path 1 ou 2 fica por sua conta. Como forma de organização recomendada, as avenidas principais podem ser o path1; as laterais que cortam elas o path2; as entradas à esquerda o path3 e/ou path4 e por aí vai.

**IMPORTANTE: QUANTO MAIS OBJETOS DE SEMÁFOROS O MAPA TIVER, MAIS PODE PESAR OU DEMORAR PRA CARREGAR. É RECOMENDÁVEL USAR OS SEMÁFOROS APENAS NOS TRAJETOS DAS ROTAS ONDE O JOGADOR VAI PASSAR, SE TRATANDO DE UM SIMULADOR... EVITE COLOCAR EM LOCAIS INACESSÍVEIS OU EM RUAS MUITO MORTAS.**

## Configurando o trigger do bloqueio para os veículos

Assim como as luzes, os semáforos têm o **trigger** que deve ser colocado no modelo 3D. Ele gera um objeto invisível que vai ficar no meio da pista onde os carros passam.

Coloque o trigger no meio da rua onde o carro deve parar. Se ele parar muito pra trás, experimente levar o objeto do trigger um pouco mais pra frente. Ou se for o caso, reduza um pouco o raio no txt, na propriedade **triggerRadius** lá no começo da seção **[trafficlight]**. Veja um exemplo de bom posicionamento dos triggers:



O nome do trigger segue a mesma estrutura das luzes, com a diferença de que em vez da cor ele vai receber a keyword `trigger` e não precisa de material, textura nem do shader additive. Por exemplo:

`_farol1_path1_trigger_`

**IMPORTANTE: CADA OBJETO DE TRIGGER DEVE SER INDIVIDUAL. NÃO JUNTE ELES COM O CTRL+J NÃO, POIS ELES SERÃO USADOS PARA PEGAR A POSIÇÃO. CASO JUNTE, A POSIÇÃO PODERÁ FICAR INCORRETA SENDO CAPTADA NO MEIO DA FORMA GEOMÉTRICA DELES.**

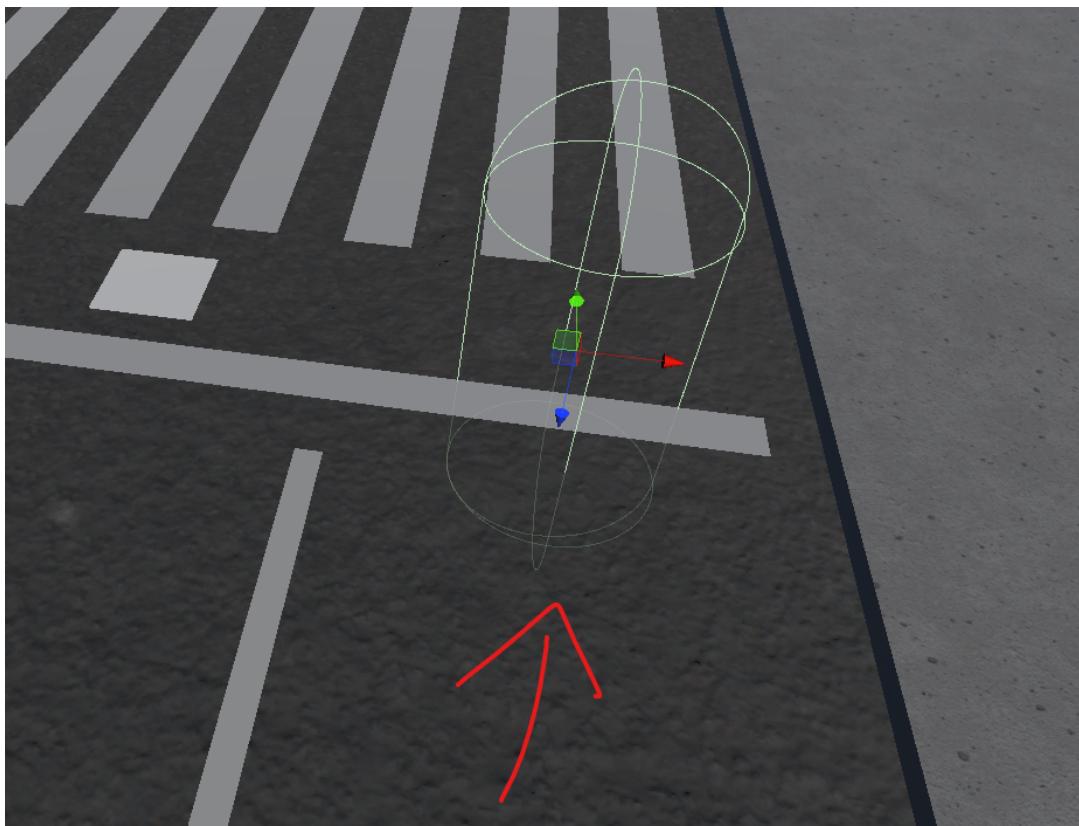
O trigger de cada path deve ficar no caminho que será bloqueado pela luz do semáforo vermelho em questão. Se a avenida principal for o path1, o farol vermelho para quem está nela terá o nome com o path1, bem como o trigger dela.

O trigger dos semáforos será configurado na engine na forma de cápsula, é o formato que se mostrou mais viável. Você vai definir o raio, que é quão “gorda” ela é, e o jogo controla a altura internamente. Um bom valor para o raio parece estar entre 1 e 2 (unidade em metros). Se for muito grande pode atrapalhar os veículos que vêm de outras direções, especialmente em cruzamentos do tipo T ou para quem vira à direita ou à esquerda. Se for muito pequeno pode ser que os carros não “enxerguem” o colisor e passem por cima. Isso também pode acontecer se ele estiver fora do centro da pista. Não precisa ter um alinhamento perfeito, mas é importante que ele esteja no meio da pista por onde passam os paths do tráfego.

Usando o modo debug nas configurações você pode ter uma indicação visual de como e onde estão os triggers. Isso será bastante útil para tentar entender o motivo de alguns carros ficarem parados ou passando no vermelho.

**DICA: EM ALGUNS LUGARES DO MAPA PODE SER ÚTIL CRIAR UM SEMÁFORO INVISÍVEL QUE USA OS TRIGGERS PORÉM SEM AS LUZES, COMO EM ALGUMAS LIGAÇÕES EM “Y” OU ENTRADAS EM RODOVIAS MOVIMENTADAS. ASSIM EVITA QUE O CARROS BATAM. PODE SER ÚTIL EM SAÍDAS DE GARAGENS E/OU TERMINAIS TAMBÉM, EMBORA NESSES CASOS MUITOS DA VIDA REAL CONTAM COM SEMÁFOROS VISÍVEIS TRADICIONAIS PARA BLOQUEAR A RUA.**

Na imagem do exemplo mostrado anteriormente, internamente na engine o trigger do semáforo ficaria assim:



O objeto do trigger será usado só para pegar a posição e em seguida descartado. Ele pode ser um triângulo ou quadrado sem textura, não precisa ser grande. Se quiser usar um material colorido no Blender apenas para poder visualizá-lo melhor durante a edição pode usar, mas isto será um dado inútil a mais para ser carregado e em seguida descartado.

## Fase 3: Luzes dos postes e ambientes

As luzes da rua e de ambientes como praças, terminais, rodoviárias etc dão uma boa imersão no cenário, especialmente na versão PC no modo ultra. No celular elas não iluminam o chão nas versões atuais, mas provavelmente este recurso será levado para a plataforma daqui uns anos, conforme os aparelhos vão ficando mais potentes. Não é uma garantia, apenas um desejo. O arquivo de configuração para os tipos de luzes tem essa estrutura:

\*luz 0001.txt - Bloco de n...

Arquivo Editar Formatar Exibir Ajuda

[streetlight]  
prefix=luz001  
alwaysOn=0

[real]  
colorR=0.8352941  
colorG=0.8352941  
colorB=0.2941177  
type=point  
range=20  
intensity=0.8

[fake]  
shader=additive  
texture=luz-rua1.png  
alwaysFaceCamera=0  
colorR=0.8382353  
colorG=0.7667387  
colorB=0.6594939  
colorA=0.5  
texScaleX=5  
texScaleY=5  
texScaleZ=5

As configurações das luzes ficam em arquivos txt na pasta **streetlights**, esta que fica dentro da pasta do tile - a mesma que contém as pastas aipeople, aivehicles, trafficlights, etc.

São configuradas dois tipos de luzes: uma luz “fake”, que é um plano com uma textura. Esse vai tanto para o PC como o celular. E tem também uma luz “real”, que é a que de fato ilumina os objetos ao seu redor e pode gerar sombras, caso ativadas nas configurações do jogo.

Os objetos no 3D do mapa serão objetos temporários só para pegar a posição. Podem ser planos pequenos sem texturas, nos locais onde as luzes forem ficar. A luz real pode ficar mais para baixo ou para o meio, não precisa ficar grudada na mesma posição do poste. Dependendo do cenário ela pode oferecer um melhor resultado estando um pouco mais para baixo do que no poste.

Cada tipo de luz diferente precisa de um txt. Em geral muitas luzes do cenário vão compartilhar o mesmo txt, por exemplo, todos os postes que tiverem a mesma lâmpada... Será necessário criar um txt diferente caso queira mudar a cor ou área de atuação da luz, intensidade etc.

A primeira seção define o básico:

### **[streetlight]**

**prefix** define o nome interno da luz no 3D. Precisa ser um nome único que não seja utilizado em nenhum outro objeto do mapa, somente para as luzes mesmo. É bom nomear algo como luz001, luzparque008 etc. Os objetos terão esse prefixo acrescentado de \_fake\_ ou \_real\_, dependendo do tipo.

**alwaysOn** define se a luz ficará sempre acesa ou não. É um valor booleano, 0 para desativar isso, 1 para ativar. Caso deixe em 0 a luz só será acesa à noite (ou com chuva). Deixando em 1 ficará ativa o tempo todo (útil para túneis ou ambientes com coberturas).

## Configurações das luzes reais

### [real]

**colorR** = cor do vermelho em RGB, de 0 a 1 (consulte o anexo da tabela de cores)

**colorG** = cor do verde em RGB, de 0 a 1

**colorB** = cor do azul em RGB, de 0 a 1

**type** define o tipo interno da luz, atualmente será sempre **point**

**range** define a distância que a luz pode atingir, algo como um raio. O valor irá depender da altura e do lugar, precisa testar caso a caso. Um valor pequeno irá iluminar somente bem perto da lâmpada, enquanto um valor maior abrange uma área maior.

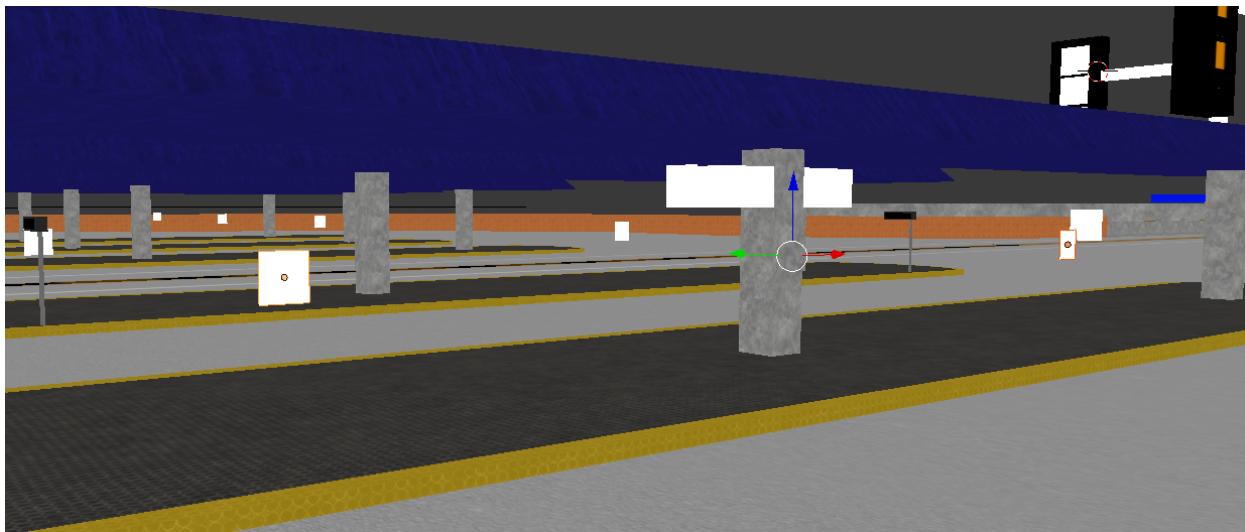
**intensity** define a intensidade, indo de 0 a 1.

As luzes reais podem ficar próximas das lâmpadas ou no meio da cena, precisa experimentar caso a caso. Se elas ficarem muito baixas pode ser ruim devido a projeção das sombras ou brilho das peças, especialmente se pegar dentro do ônibus enquanto ele passar.

Estes planos servem para pegar a posição para criar a luz dentro do jogo. Os nomes deles devem ser assim:

**\_nome definido no prefix\_real\_**

Exemplo: **\_luz001\_real\_**



## Configurações das luzes fakes

As luzes fakes são uma textura adicionada num objeto plano, quadrado, dupla face, que será criado na posição indicada pelo `_nomedaluz_fake_`. Geralmente este objeto terá seu centro sobreposto ao objeto visual da luz do poste.

### [fake]

**shader** define o shader para esta luz. Nas versões atuais use apenas o **additive**.

**texture** define a textura da luz fake. Ela precisa estar na pasta de texturas do mapa.

**alwaysFaceCamera** pode ter o valor 0 (desativado) ou 1 (ativado). Se ativado, a textura irá girar para sempre apontar para a câmera, algo perfeito para simular as luzes com efeitos redondos (a grande maioria dos postes). Deixe 0 nesse valor caso queira que o plano fique fixo. Isso pode ser útil em 1 para as luzes dos postes da rua ficarem parecidas com as dos mapas nativos. E pode ser útil em 0 para luzes posicionadas viradas para baixo em alguma cobertura, como luzes internas de terminais ou painéis luminosos.

**colorR** cor da luz referente ao vermelho em RGB, de 0 a 1 (consulte o anexo sobre as cores)

**colorG** cor referente ao verde

**colorB** cor referente ao azul

**colorA** canal alpha da cor, nesse caso da sobreposição da textura o padrão é 0.5 (na maioria dos outros elementos o padrão é 1).

**texScaleX** escala do objeto da textura no X (o padrão é 10, altere isso para mudar o tamanho).

**texScaleY** escala do objeto da textura no Y

**texScaleZ** escala do objeto da textura no Z

**rotX**, **rotY** e **rotZ** definem a rotação nos respectivos eixos. Item opcional, é ideal para luzes planas no teto onde isso pode ser 90 ou -90 para ficarem deitadas e não na vertical. A rotação será ignorada se **alwaysFaceCamera** for 1, visto que naquele modo ela sempre rotaciona para apontar para a câmera.

Os parâmetros de cor aqui vão alterar a iluminação da textura. A textura pode ter o fundo preto (veja os mapas de exemplo), ele será eliminado com o shader **additive**. Se preferir não configurar a cor no txt tudo bem, você pode pintar a textura na cor desejada em vez de branca.

## Observação sobre as luzes

A maioria dos elementos de luz terá os dois objetos: real e fake. Eles precisam ser independentes. Não use `_real_fake_` no mesmo objeto porque não vai funcionar, um poderá se sobrepor ao outro.

Ao duplicar as luzes não é necessário duplicar os txt, apenas duplique os objetos no 3D mesmo.

Caso tenha algum objeto iluminado que vai ficar sempre aceso você pode usar o `_emissive_` direto no objeto, sem precisar nomear ele como luz. Aí coloque uma luz real nas proximidades para iluminar o cenário.

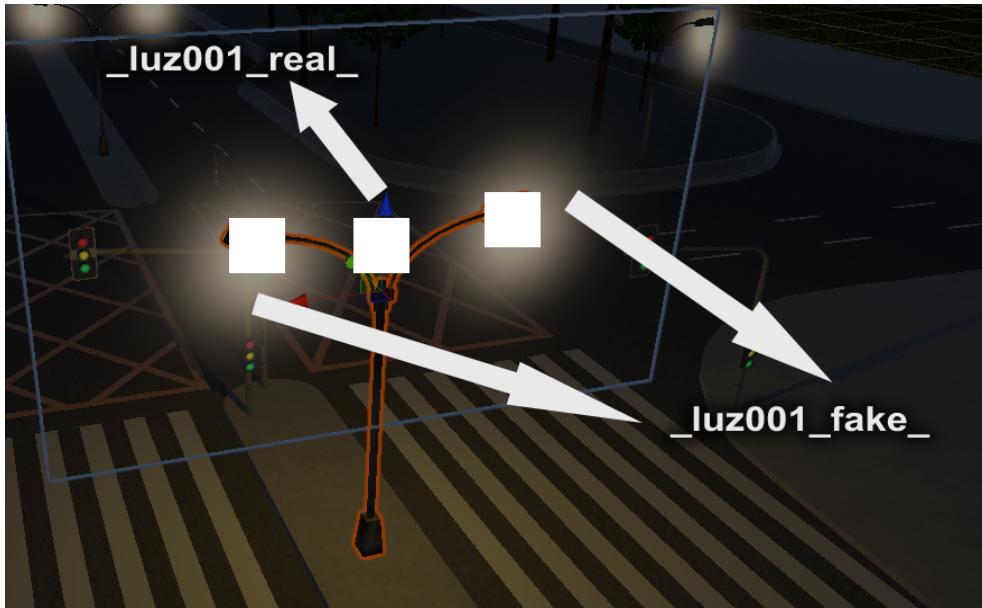
As luzes reais não podem ficar dentro de objetos opacos ou atrás de muros, caso contrário podem não iluminar a via ou bugar as sombras. No caso dos postes, a luz real precisa ficar um pouco abaixo da peça visível do 3D da lâmpada. Se estiver dentro dele vai bugar a sombra e pode não iluminar bem o que deveria.

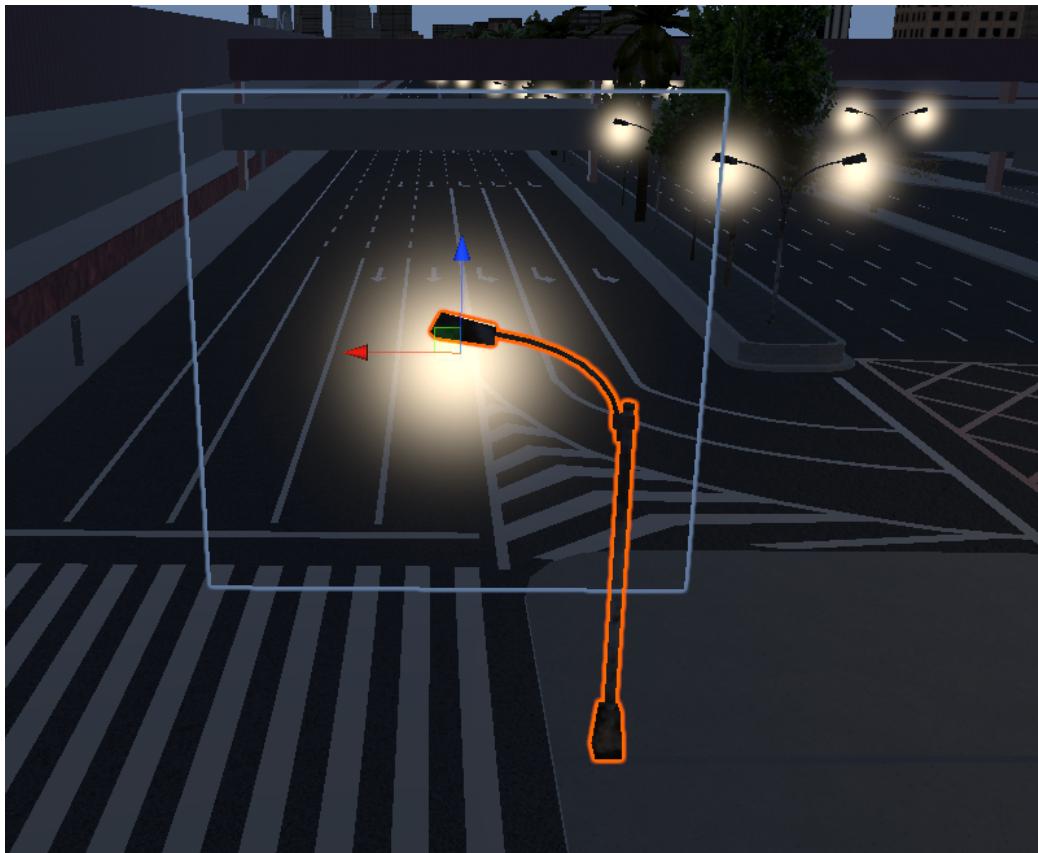
Adicionar uma grande quantidade de luzes reais pode deixar o jogo bem pesado, especialmente se o range delas for muito alto, de forma que uma área seria iluminada por várias luzes ao mesmo tempo. Isso faz a engine renderizar os mesmos objetos várias e várias vezes, prejudicando a performance. É bom não abusar das luzes reais, dando um espaçamento maior entre elas, ou reduzindo o range. Caso uma rua tenha dois postes um de cada lado, pode ser bom usar uma luz real só no meio deles para deixar mais leve (como por exemplo, deixe a luz real no meio da rua no alto, não no poste). Somente as luzes fakes poderiam ficar em cada um dos postes para poder aparecer o efeito ao redor da lâmpada.

Seguem agora alguns exemplos de boas práticas para as luzes.



Este tipo de poste usa duas luzes. Se fôssemos colocar duas luzes reais poderia ficar pesado por estarem bem próximas. Então foram usadas duas luzes fakes nas pontas dos modelos das lâmpadas, e uma luz real única no meio.



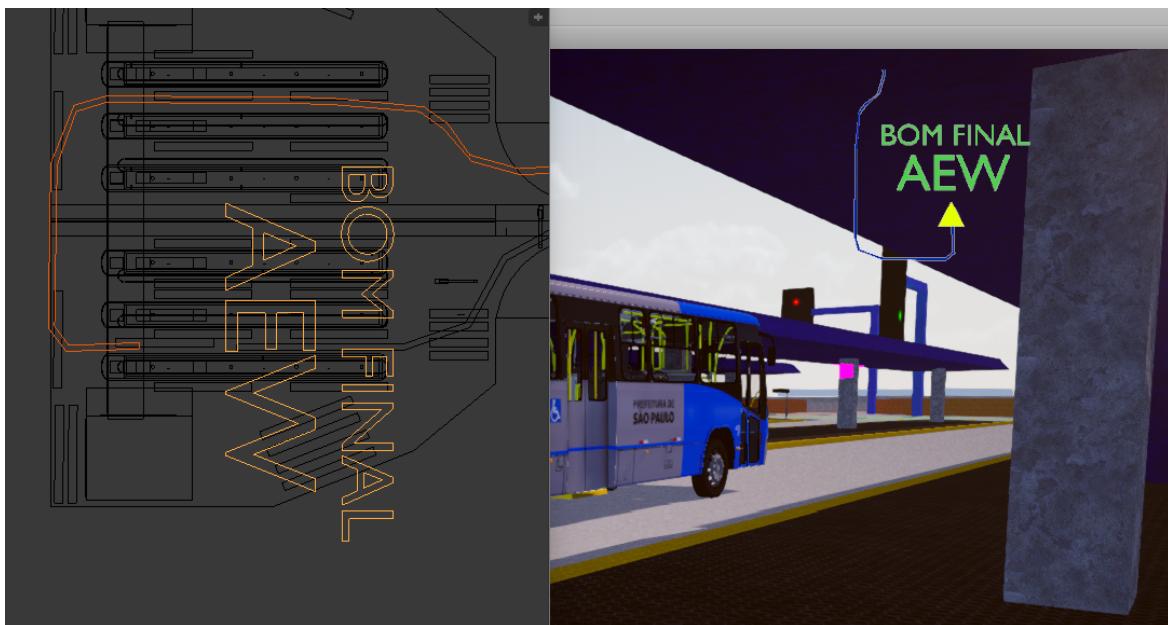


Já neste poste só tem uma luz, então a luz real pode ficar praticamente junto à luz fake, somente um pouco mais pra baixo e pra fora do 3D pra não bugar as sombras.

DICA: ALGUMAS CIDADES USAM LUZES AMARELADAS OU ALARANJADAS NAS VIAS PÚBLICAS. USAR ELAS EM VEZ DE LUZES TOTALMENTE BRANCAS DÁ UM EFEITO BEM BONITO! ALTERNAR ALGUNS LOCAIS COM LUZES BRANCAS PODE SER INTERESSANTE TAMBÉM.

## Fase 3: GPS

O GPS é basicamente um mesh normal por cima do trajeto, pode ficar no nível da rua ou levemente flutuando. Ele não pode ter colisor (por motivos óbvios). A cor pode ser definida por uma textura como num mesh normal. Durante a execução do jogo ele fica escondido da câmera principal.



É possível adicionar outros objetos auxiliares, como indicações de pontos de parada, setas para facilitar a saída de uma rodovia ou rotatória, mensagens de texto etc. Só tome cuidado com o peso caso lote de indicadores. Caso o mesh fique muito grande em rotas de vários km, pode ser interessante dividir ele em pedaços menores, otimizando a renderização. Na engine, se um pedaço do objeto aparecer ele precisa ser processado por inteiro. Então convém dividir o mesh do GPS pelo menos em dois ou três pedaços. Muitos pedaços pequenos também podem ser ruins, já que aumentam o tanto de objetos no total. Provavelmente dividindo o mesh do GPS entre 2 a 10 partes dependendo do tamanho da rota já basta.

Basicamente o objeto do GPS precisa ter o nome no seguinte formato:

**\_gps\_NOME DO ENTRYPPOINT\_**

Por exemplo:

`_gps_Linha 1 Ida_`

`_gps_309T-22 TP_`

`_gps_CARGA NORTE_`

Nem todos os entrypoints precisam obrigatoriamente ter um objeto desses. Letreiros como RESERVADO, GARAGEM etc normalmente dispensam isso.

Os outros pedaços ou indicadores podem ter mais textos no final do nome, não tem problema. Aqueles .001, .002 etc que o Blender adiciona ao duplicar um objeto não vão interferir desde que o esquema **\_gps\_nome\_** esteja presente.

**SE O MAPA TIVER MUITAS LINHAS, TENTE DEIXAR CADA MESH DO GPS O MAIS LEVE POSSÍVEL, POIS TODOS FICARÃO CARREGADOS JUNTO COM O MAPA. É RECOMENDÁVEL COMEÇAR COM UM PLANO E IR DANDO “EXTRUDE” NAS BORDAS PARA FAZER A MALHA. PARA EVITAR TRAVAMENTOS DURANTE O GAMEPLAY OS MODELOS 3D NÃO SÃO CARREGADOS DEPOIS QUE O JOGADOR JÁ ESTÁ NO CENÁRIO, TUDO É FEITO NO INÍCIO.**

Note que se o nome usar acentos ou caracteres especiais pode ser que não funcione, pois a engine pode ter dificuldade ao tentar localizá-los na hierarquia dos objetos carregados. Vale as recomendações de sempre nas nomenclaturas dos itens internos: apenas letras de a a z do alfabeto inglês, números de 0 a 9 e os caracteres underscore (sublinhado, underline) ou hífen. A barra é usada para separar caminhos de arquivos e pastas, ela **não** pode ser usada nos entrypoints também. Caso seu entrypoint tenha acentos, é sugerido remover eles na configuração dos pontos de entrada nos respectivos txt.

Como a câmera do GPS renderiza um “mundo separado” sem sol, os objetos do GPS usarão o shader similar ao `emissive` automaticamente, caso contrário poderiam ser de difícil visibilidade.

# Comandos e nomes especiais nos modelos 3D

## Peças transparentes, para grades, árvores

Peças que precisam ter uma transparência podem ser nomeadas com `_transparent_`. A palavra chave pode estar antes ou depois do nome do objeto, é indiferente. Por exemplo, `placa_transparent_` e `_transparent_placa` têm o mesmo efeito.

Para que a transparência funcione de fato, a textura do material usado no objeto precisa ter o canal alpha com algum valor transparente. Normalmente você pode ajustar isso nos programas gráficos; isto foge ao escopo deste manual.

No momento este comando utiliza um shader simplificado de transparência para ser leve. Ele pode não emitir sombras. Futuramente poderão existir outros comandos relacionados.

## Peças com colisor, para ruas, calçadas, plataformas, pisos

Todas as peças que precisam ter colisor podem ter o nome `_gencol_` junto ao nome do objeto. Tendo esta palavra chave o jogo tentará criar um colisor para a peça usando as funções internas da engine para isso.

Para fins de otimização, utilize este comando somente quando necessário! Se tiver muitos objetos inúteis ou complexos com colisor, o jogo poderá rodar travando ou ficar bem mais pesado.

Recomendamos usar colisores apenas em peças de chão, ruas, plataformas de terminais, colunas onde o jogador possa bater, etc. Evite colocar em casas, prédios, muros detalhados etc, isso só fará ficar mais pesado à toa.

IMPORTANTE: A GERAÇÃO DO COLISOR É UMA FUNÇÃO DA ENGINE QUE NÃO TEMOS ACESSO. FUTURAS VERSÕES DO JOGO PODERÃO QUEBRAR A COMPATIBILIDADE COM OS COLISORES ATUAIS POR CONTA DISSO, SENDO NECESSÁRIO, QUEM SABE, UMA ATUALIZAÇÃO NOS MAPAS PARA ALGUM NOVO PADRÃO. PARA MINIMIZAR OS IMPACTOS DE ATUALIZAÇÕES FUTURAS NOS COLISORES, EVITE AO MÁXIMO GERAR COLISORES COMPLEXOS, COM MUITOS POLÍGONOS. SE FOR NECESSÁRIO, MANTENHA A PEÇA VISUAL SEM COLISOR, GERANDO UM COLISOR INVISÍVEL COM UMA FORMA GEOMÉTRICA SIMPLIFICADA, NO COMANDO A SEGUIR.

MUITO, ABSOLUTAMENTE MUITO IMPORTANTE: COLISORES GRANDES DEMAIS PODEM SER PESADOS PARA PROCESSAR OU ATÉ MESMO PROBLEMÁTICOS, CAUSANDO INSTABILIDADES NOS CÁLCULOS DA FÍSICA DO JOGO. SE A SUA RUA FOR UM MODELO ÚNICO GRANDE, DIVIDA ELA EM PARTES COM ALGUMAS DEZENAS DE METROS (OU ALGUMAS POUCAS CENTENAS). NÃO FAÇA MESHES QUILOMÉTRICOS COM COLISOR, POIS PODERÃO DAR PROBLEMAS MAIS CEDO OU MAIS TARDE.

## Peças invisíveis com colisor, para bloqueios, pisos complementares

Às vezes você quer colocar um colisor apenas para evitar que o jogador ultrapasse determinada área, mas não quer deixar o local “feio” com um bloqueio visível. Nesse caso você pode usar, em combinação com o `_gencol_`, o comando `_invisible_`. Ele basicamente remove o renderizador do objeto. Nesse caso nem precisa ter material ou textura, apenas o mesh mesmo. Por exemplo: `barreira1blablabla_gencol_invisible_`.

Isto também é útil para colisores em modelos complexos. Por exemplo, um poste detalhado. É desnecessário colocar um colisor envolvendo todo o poste usando o mesmo modelo visual, já que isso aumentaria muito os cálculos da física que a engine precisa fazer. Quanto mais polígonos nos colisores, mais peso!

Nesse caso você pode deixar o modelo do poste sem colisor nenhum, e colocar um colisor simplificado somente no corpo dele com o comando `_gencol_invisible_`, usando um cubo ou um cilindro de poucas faces (6, 8 etc, de preferência não muitas).

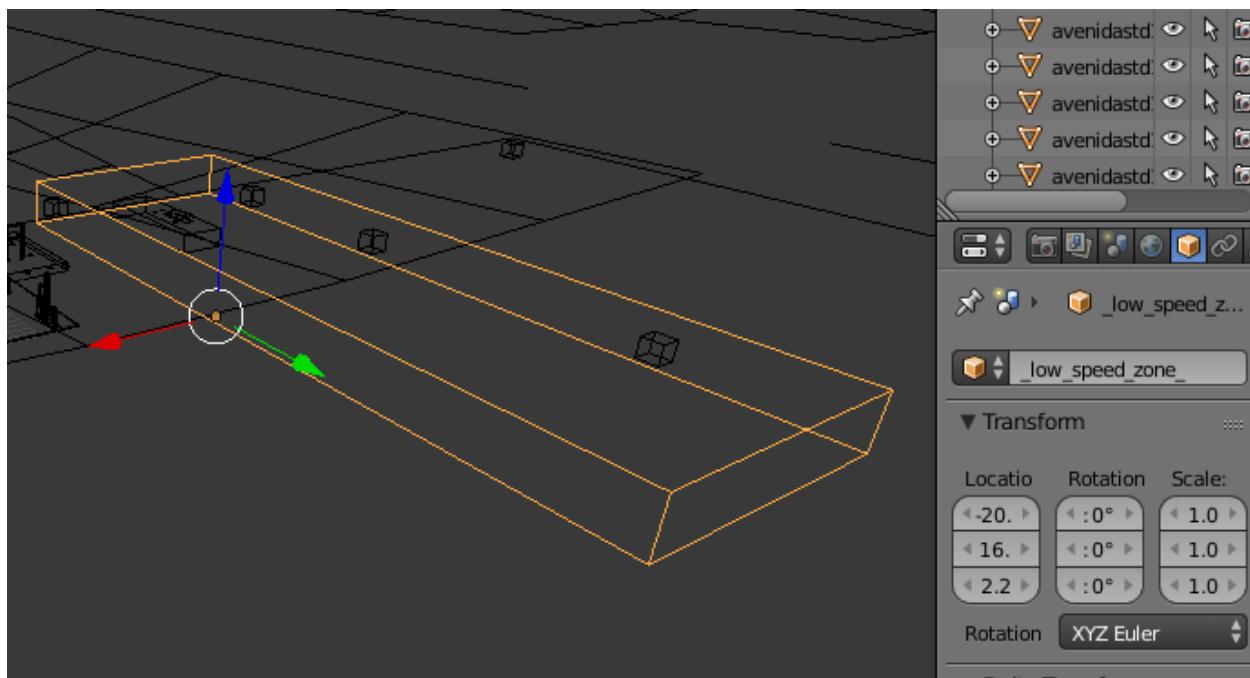
## Peças sempre acesas, como painéis luminosos de propagandas

O comando `_emissive_` altera o shader do material para deixá-lo sempre aceso. É útil para painéis luminosos, como propagandas em pontos de ônibus e algumas fachadas de lojas, reflexos em tartarugas, placas eletrônicas de radares ou terminais etc.

A fase 3 adicionou também o comando `_additive_` para o shader additive, recomendável para os semáforos. Este comando usa uma textura de fundo preto, opcionalmente com gradiente em torno das luzes. A parte preta é removida e a parte branca aparece acesa.

## Indicador de área de terminal ou ponto final, para não reclamarem da velocidade baixa

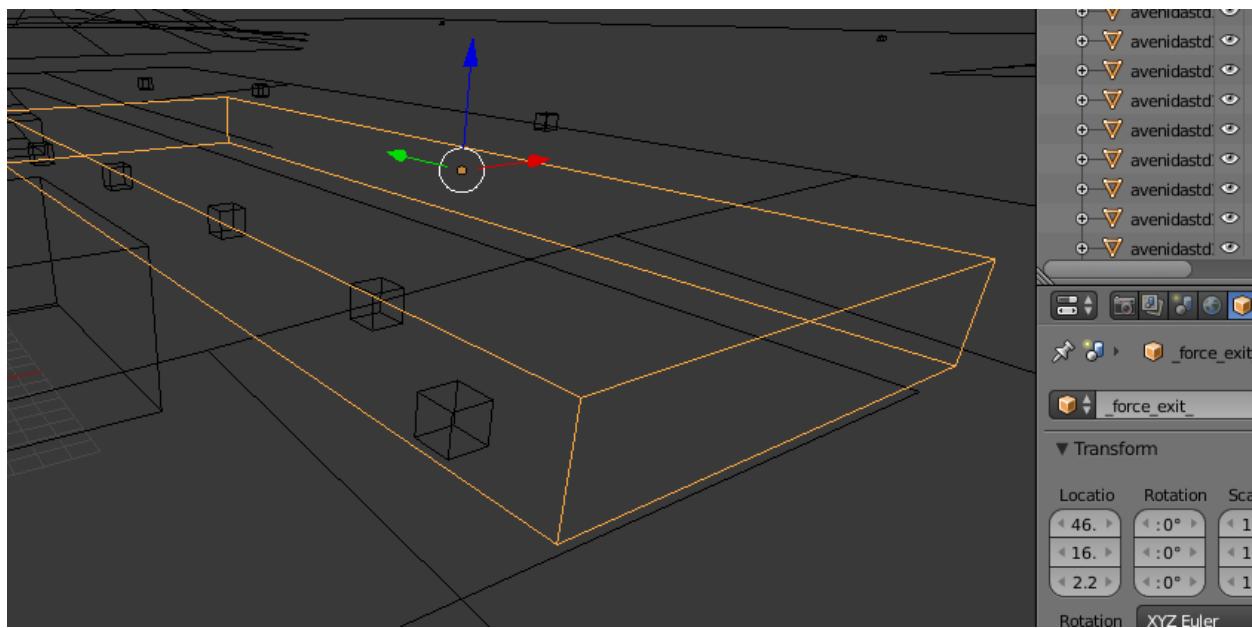
Quando o motorista anda devagar os passageiros podem reclamar da velocidade baixa. Em alguns casos é necessário manter velocidade baixa na vida real, como dentro dos terminais. Para evitar reclamações de lerdeza dentro dos terminais, você pode adicionar um objeto cúbico englobando toda a área do terminal com o nome `_low_speed_zone_`.



Ele vai ofuscar sua visão no 3D, mas basta movê-lo para outra camada para não ficar vendo durante a edição. Esse objeto será um trigger invisível dentro do jogo, não precisa ter material nem textura.

## Indicador de ponto final, para todo mundo desembarcar

Quando estiver chegando perto do ponto final, um pouco antes de entrar na rua dele, você pode colocar um cubo com o nome **\_force\_exit\_** para forçar todo mundo a descer do ônibus.



Assim como o cubo do low speed zone, este não precisa de material nem textura.

**IMPORTANTE: EM FUTURAS VERSÕES DO JOGO AS ROTAS PODERÃO SER DEFINIDAS A PARTIR DE UMA LISTA DE PONTOS. NESSE CASO O JOGO PODERÁ FAZER TODO MUNDO DESCER NO ÚLTIMO PONTO DA LISTA, SEM PRECISAR DISTO.**

### Fase 3: objetos aleatórios que podem estar ou não no lugar

A fase 3 adiciona o comando `_rand_` para o nome dos objetos, seguido de um número entre os fornecidos abaixo. Ele permite que o objeto em questão desapareça mediante uma probabilidade/sorteio. É como se ele tivesse uma certa porcentagem de chance de estar presente ou não. Pode ser útil para ônibus e veículos estacionados e objetos de detalhes não-fixos (bicicletas, patinetes, animais etc).

**ESTE COMANDO REQUER QUE O OBJETO ESTEJA NUMA PEÇA ÚNICA. SÓ VAI FUNCIONAR NESSA FASE COM UM OBJETO DE UMA ÚNICA TEXTURA. CADA OBJETO TERÁ UM SORTEIO INDEPENDENTE DOS DEMAIS.**

Por exemplo, `_rand_30_` faz o objeto ter 30% de chance de aparecer. `_rand_5_` apenas 5%, tornando o objeto mais raro. `_rand_99_` deixa ele quase sempre aparecendo, mas eventualmente não vai estar lá.

O sorteio será feito com números de 0 a 100. Para fins de otimização na busca pelas peças ao carregar o mapa, nem todos os números podem ser usados, assim ficam menos verificações rodando. Estes são os números que podem ser usados no `_rand_`:

1, 3, 5, 10, 20, 30, 40, 50, 60, 70, 80, 90, 95, 99

Eles dão uma boa variedade para a grande maioria das situações.

Por ser um evento aleatório dependendo do sistema de sorteio da engine, não é possível garantir que sempre vai bater em poucas execuções. Pode ser necessário centenas ou milhares de carregamentos para começar a observar os resultados.

## Detalhes extras, para deixar o mapa mais leve

A potência dos celulares é muito variável, ainda mais lidando com o Brasil: tem muito celular popular fraco ou antigo. Se detalhar demais... O mapa pode ficar pesado e ninguém conseguir rodar... Infelizmente achar um balanceamento entre qualidade e peso é algo bem complicado.

Tentando ajudar nisso o jogo tem a opção “Detalhes extras” na seleção de mapas, que cada jogador pode marcar ou desmarcar quando quiser. O ideal é que alguns modelos de cenário que não são tão importantes sejam colocados como um detalhe extra, podendo ser desativados caso necessário.

Os mods de mapas também suportam isso!

Nomeie o arquivo 3D dos objetos menos importante com um **\_det1** no final do nome, antes da extensão. Por exemplo, **mapaprincipal.3ds** para o mapa em si e **maiscoisas\_det1.3ds** para os objetos de detalhes extras. Note que isso vale para o arquivo exportado que fica na pasta, não é o nome do objeto dentro do Blender não!

Esses que terminarem em **\_det1.3ds** só vão ser carregados se o usuário marcar a opção de detalhes extras, podendo salvar tanto processamento gráfico (menos triângulos e texturas) como memória RAM e de vídeo.

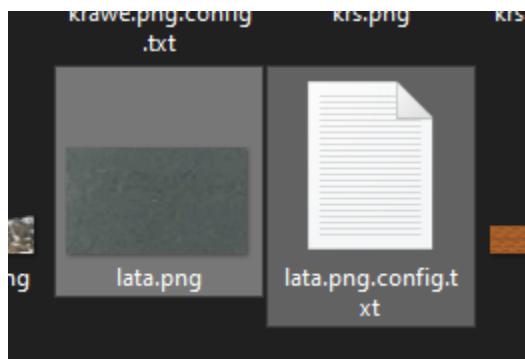
**IMPORTANTE: NO MOMENTO PODE NÃO SER SUPORTADO MANTER OS CAMINHOS DO TRÁFEGO, PASSAGEIROS ETC NOS DETALHES EXTRAS. USE APENAS PARA MODELOS ESTÁTICOS DE CENÁRIO, COMO ÁRVORES, ALGUNS PRÉDIOS, TELEFONES PÚBLICOS, OUTDOORS, BANCOS EM PRAÇAS ETC.**

## Configuração das texturas cintilantes no chão e muros, telhados

Algumas texturas que preenchem uma área visualmente grande podem exigir um tratamento especial, senão elas ficam piscando ou cintilando dentro do jogo, parecendo que estão “vivas” ou com “glitter”.

Para fins de otimização o Proton não faz esse tratamento automaticamente, já que na maioria das texturas isso não faz diferença. É necessário especificar manualmente quais são as texturas em você quer aplicar o recurso. Normalmente isto só deve ser usado em texturas de chão, como o asfalto, calçadas, gramas, terreno, ou muros, coberturas grandes etc.

Basicamente, crie um arquivo txt vazio com o mesmo nome da textura, na mesma pasta dela, adicionando ao final do nome o texto **.config.txt**. Por exemplo:



Para a textura asfalto.png, por exemplo, o nome do txt deveria ser asfalto.png.config.txt.

**PREFIRA MARCAR PARA VER AS EXTENSÕES DOS TIPOS DE ARQUIVOS CONHECIDOS NAS OPÇÕES DE PASTAS DO SEU SISTEMA OPERACIONAL, ISSO AJUDA A EVITAR ERROS AQUI.**

Caso a textura fique cintilando durante a movimentação do ônibus, fazer isso deve ajudar a resolver. Novamente, a recomendação: não abuse disto, use só quando necessário nas texturas de chão, muros ou coberturas grandes que ficarem piscando. Este comando faz a textura consumir um pouco mais de RAM e eventualmente de processamento gráfico. Por isso se usar em todas de uma vez poderia ser pior.

## Extras - Dicas de otimização

No geral, quanto menos triângulos e texturas, melhor.

Quanto menores forem as texturas, melhor também. Muitas vezes ao usar uma textura que se repete no objeto (tiled ou seamless) dá para deixá-la em 256x256 ou 512x512 pixels, para melhor performance. Não precisa ser 1024 ou maior se o nível de detalhes não for fazer muita diferença, dependendo do tamanho do objeto. Use texturas maiores quando for realmente necessário destacar algo pequeno nelas, como um texto, por exemplo, nas skins dos ônibus.

**IMPORTANTE: NUNCA USE TEXTURAS COM DIMENSÕES ACIMA DE 2048 PIXELS PARA MAPAS PARA CELulares! A MAIORIA DAS GPUS MOBILE NÃO ACEITAM ESTAS TEXTURAS, PODENDO CAUSAR EFEITOS COLATERAIS OU CRASHES ALEATÓRIOS NO JOGO. NÃO SE TRATA DE CELulares BARATOS, MAS DE GPUS MOBILE MESMO, ACONTECE ATÉ EM ALGUNS TOPO DE LINHA.**

Se tiver muitos objetos com texturas grandes na cena, como os ônibus estacionados, algumas placas, outdoors etc, prefira deixá-los na camada de detalhes extras. Dessa forma eles nem serão carregados para os usuários que desmarcarem a opção dos detalhes extras.

Se for viável, faça duas versões do mapa: uma normal, completa, para PC; e uma com as texturas redimensionadas para os celulares. Dependendo do caso um mapa só para os dois já basta. Se o mapa for muito pesado, alguns usuários costumam editar as texturas para ficarem mais leves e compartilham com os outros. Evite ficar zangado caso isso ocorra, a arquitetura aberta dos mods do Proton não impõe restrições, permitindo que os mods possam ser acessíveis ao maior número de pessoas possível. Se você não "otimizar" seu mapa, provavelmente alguém vai tentar. Ninguém tem como ter controle sobre isso.

Alguns objetos podem usar uma textura de cor sólida, sem detalhes. Nesse caso pode-se usar uma textura de 4x4 pixels, bem pequena mesmo, só com a cor. É um baita desperdício criar texturas grandes para ter uma cor só, visto que não há detalhes nelas que precisam ser diferenciados nesse caso.

Sobre os colisores, como já foi dito no comando `_gencol_`, evite criar colisores muito grandes ou complexos. Eles aumentam radicalmente a quantidade de cálculos matemáticos que a engine física do jogo precisa realizar para o processamento. Se o seu mapa for plano, prefira utilizar um ou mais colisores planos invisíveis na mesma altura das ruas, em vez de ficar criando colisores em todas elas.

Objetos de construções como garagens, terminais etc, não precisam ter colisores em todos os vértices. Prefira utilizar apenas no chão e nas plataformas, calçadas, barreiras etc, por onde o ônibus pode naturalmente passar. Não precisa colocar o comando de colisão no objeto todo com as telhas, colunas e vigas etc, seriam muitos cálculos a mais completamente desnecessários.

Um alto número de colisores complexos fará o processo técnico do "floating origin" ficar mais demorado, eventualmente aumentando a travada que o jogo dá ao parar o veículo depois de ter dirigido por alguns km.

Usar um objeto só com o mapa inteiro pode não ser recomendável, bem como objetos complexos muito grandes, como muros com 1 ou 2km. Eles podem bugar a sombra ou a iluminação caso os vértices sejam muito afastados uns dos outros. Todavia, se usar milhares de objetos pequenos, outro efeito ruim pode aparecer: mais peso para processar todos eles. Não existe um número mágico que valha para todos, o ideal é testar nos aparelhos até achar um bom equilíbrio entre performance e qualidade. Recomendamos dividir objetos grandes em partes menores caso passem de 40, 50, 100 metros... Especialmente se forem objetos complexos. Quando uma pequena ponta do objeto precisa aparecer na tela, todo ele é processado pela engine, o que pode causar um peso desnecessário se ele for grande demais. Isso pode causar uma perda de performance considerável porque a engine precisa processar um monte de vértices para no final só mostrar alguns deles. Por isso dividir as peças quilométricas de ruas e muros, telhados etc em partes menores pode ajudar a reduzir a quantidade de triângulos que cada cena precisa processar a cada momento.

Evite utilizar vários materiais num mesmo objeto. Quando tem dois ou três materiais, o objeto precisa ser desenhado em várias etapas na engine, uma para cada material. Isso aumenta o

tempo de produção de cada frame, fazendo cair os fps. Prefira separar por materiais. Você pode parentear os objetos no Blender para mantê-los juntos ou relacionados (CTRL + P), é melhor do que um objeto só com tudo junto (CTRL + J). Por exemplo, num terminal que usa vários materiais, deixe separado por material cada peça, de forma que cada uma fique só com o seu material associado. Plataforma, piso, telhado, etc. Tende a ser mais eficiente do que deixar o terminal inteiro num objeto só com 5 ou 10 materiais.

Combine objetos pequenos que estejam próximos caso usem o mesmo material. Isso também ajuda a reduzir o tanto de trabalho que o processador e a placa de vídeo precisam fazer. Um objeto grande é ruim, um monte de objetos pequenos também é ruim. Os objetos pequenos que usam o mesmo material podem ser combinados caso estejam próximos fisicamente (CTRL + J no Blender), assim são renderizados todos juntos. Isso é válido para pedras, árvores ou até mesmo prédios. Mas tome cuidado: para que isso funcione bem, eles precisam ter um único material e estarem próximos uns dos outros no mapa. Caso contrário você enfrentaria o velho problema de ter um objeto grande, aumentando o número de polígonos inúteis que precisariam ser processados e a seguir descartados quando apenas uma parte do objeto estivesse visível.

Se isso parece muito confuso, tente isso: combine de 3 a 5 ou 10 árvores que estejam próximas umas das outras, caso usem o mesmo material. Dessa forma elas serão renderizadas todas juntas, poupando recursos de processamento quando a engine precisar pegar uma lista de todos os objetos que estão no alcance da câmera que devem ser desenhados na tela.

Se o mapa for muito grande, pode ocorrer perda da precisão das coordenadas de posição ainda dentro do Blender, quanto mais você se afastar da origem (o ponto 0,0,0). Você pode dar uma aliviada nisso expandindo para as duas pontas do eixo, tanto positivo como negativo, mantendo o mapa dentro do intervalo de -10 a 10km, por exemplo. Tende a ser mais eficiente do que sair do 0 aos 20km.

## Quais os impactos se fizer um mapa muito grande?

O sistema de mapas do Proton nessa fase não é adequado para mapas grandes. Não sabemos se isso será suportado no futuro. Desejamos, mas tecnicamente é muito difícil fazer. Um mapa enorme precisa ser dividido em pedaços, tendo os pedaços carregados e descarregados conforme o jogador se movimenta no cenário. O problema disso é o impacto na performance: cada vez que o jogo vai carregar e integrar os objetos novos na cena dá uma travada. Essa travada é cada vez maior conforme a complexidade dos detalhes aumentarem. Não queremos que o jogo fique travando por isso, então nessa fase o mapa é carregado por inteiro, para não ter o impacto negativo do carregamento no meio do cenário. Ele pode até travar por outros motivos. Por exemplo, os spawners de pessoas, carros, passageiros etc precisam instanciar elementos e isso tem um impacto no thread principal do jogo. Todavia o lag massivo por carregamento de cenário vamos evitar.

Consequentemente, se o mapa for de muitos quilômetros, a quantidade de objetos, polígonos e texturas tende a ser muito alta. Isso deixaria o mapa inteiro pesado. Ou pior, faria o jogo crashar por falta de RAM, especialmente no Android ou em PCs de 32-bit.

Não dá para determinar um tamanho máximo fixo, depende dos detalhes e tudo mais. Um fato a considerar são os objetos dos paths de tráfego, especialmente os spawners: eles precisam ficar rodando um código a cada temporada (o intervalo de spawn). Esse código é distribuído ao longo dos frames, mas quanto mais objetos com isSpawner tiver, mais vezes esse código vai rodar quase ao mesmo tempo... O que poderia fazer o jogo ficar travando, tanto nos celulares como nos PCs. Se o mapa tiver um monte de pontos do tráfego e tiver várias dezenas de quilômetros talvez este seja o motivo das travadas, principalmente se forem vias de várias faixas.

Ainda assim é possível ter rotas boas, com duração de 20, 30, 40 minutos: o antigo mapa do corredor da linha 4520 era carregado de uma vez só antes de ser integrado ao Aricanduva. Ele tinha três faixas de cada lado, tendo inúmeros pontos de spawner, mesmo assim era utilizável com boa performance nos celulares. Vale fazer e testar, estando ciente da limitação.

Caso seu mapa for ter muitas linhas, realmente recomendamos separar ele em mapas menores, criando vários .map.txt para cada região. Eles até podem compartilhar a pasta principal do mapa, para evitar duplicar todas as texturas. Serão mais eficientes se o usuário só precisar carregar a área em que realmente for jogar.

## A complexa questão de mapas abertos na Unity

Um efeito colateral de fazer linhas longas em mapas na Unity atualmente é a perda de precisão dos números, e ainda não existe uma solução para essa limitação. A Unity usa números de ponto flutuante de 32-bit para armazenar dados das posições, que podem guardar apenas 7 dígitos de precisão. Se você dirigir por mais de 1000 unidades (1000 metros ou 1km) a partir da origem da cena (o ponto onde  $x=0, y=0, z=0$ ), sobram apenas três vagas para as casas decimais. Depois de 10000 unidades (10000 metros ou 10km) sobram apenas duas vagas. As coisas se complicam pra valer aqui: a sombra fica piscando, objetos pequenos aparecem tremendo fora do lugar original, etc... Essa engine não tem um bom suporte a mundos abertos grandes no seu modelo clássico de programação.

Não existe nenhuma solução mágica, só gambiarras. O Proton Bus move tudo na cena para manter a câmera perto do ponto 0,0,0, dessa forma a engine nunca ficará sem vagas para a parte decimal das posições. Esse é o processo de reset da origem ou floating origin. Mas é uma operação pesada. Para mover todos os objetos, especialmente colisores, a engine física precisa refazer um monte de cálculos porque todas as posições mudam de uma vez. Traduzindo para o mundo real: o jogo irá travar por um momento. Geralmente uma fração de segundo. Mas isso não é bom. Evitamos fazer esse processo com o player se movendo, para evitar quebrar a imersão da experiência. Isso será feito quando o player parar o ônibus. Então se você dirigir por alguns quilômetros sem parar o veículo por um momento, irá experimentar estas coisas ruins.

Vamos continuar pesquisando para tentar resolver essa limitação, mas dependemos de alterações na engine em si, uma caixa preta que não temos acesso. Outras engines podem aplicar outras soluções, mas na nossa realidade atual não há muito mais a fazer.

Desejamos que um dia o Proton suporte mapas realmente grandes, com o streaming de cenário. Mas não sabemos se isso será possível. Não será um recurso prometido. Se der certo, ótimo. Se não der, vamos focar no sistema atual que já está dando certo para a maioria.

## Ferramentas de Debug

Você pode ativar as ferramentas de debug para visualização de algumas coisas que geralmente são invisíveis, relacionadas aos mods. Nas versões em torno da 252 ou 253 estas ferramentas estão nas telas de seleção dos ônibus e mapas, mas futuramente poderá ser necessário ativar algum item na tela Extras para liberá-las, evitando que os jogadores normais se percam nelas.

Basicamente aparecerão contornos em volta dos colisores especiais dos pontos, triggers, etc, bem como nos caminhos dos pedestres, carros e trens. Isso ajudará a identificar potenciais problemas que possam passar despercebidos no Blender, como pessoas flutuando ou andando em alguma direção incorreta.

**Nunca use imagens com dimensões acima de 2048 pixels de largura ou altura para os celulares atuais!**



Novamente, reafirmamos, por favor: tanto para ônibus como para mapas, não use imagens com dimensões superiores a 2048 pixels para mapas de celular!

Até mesmo no PC estas imagens normalmente não são necessárias, exceto em casos de skins ou algumas placas com textos que precisam aparecer legíveis.

Nas versões atuais dos mapas os usuários podem marcar uma opção para liberar o carregamento das texturas grandes, mas isso traz tantos problemas de suporte que talvez um dia será removido. Então o jeito é evitar mesmo. No jogo compilado para Android a própria Unity redimensiona as texturas para o máximo de 2048 pixels, caso sejam maiores. Nos mods fica a critério dos criadores e dos usuários. Um fato é que não podemos ficar recebendo reclamações e casos de suporte por um "problema" que não é nosso, pois isso atrapalha o fluxo do

desenvolvimento das demais coisas. Vários glitches e crashes aleatórios eram desconhecidos, demorou para vermos que a causa era o simples fato de estar com alguma textura grande carregada nos celulares.

Neste artigo há uma explicação:

<http://www.busmods.com/blog/static/1571653273imagens-grandes/>

E aqui um relato da descoberta:

<http://blog.protonbus.com.br/2019/10/importante-evitem-mods-com-texturas.html>

Contamos com sua compreensão e colaboração.

Mesmo quando não dá glitch nem crash, as texturas grandes têm a desvantagem de ocupar quase toda a largura de banda entre a GPU e a CPU ou armazenamento ou a RAM. Há um limite físico de transferência que varia de aparelho para aparelho, como se fosse uma conexão de internet ou um cano com determinada vazão. Lotar de texturas grandes faz o sistema ter que esperar mais tempo para processá-las cada vez que o objeto vai aparecer na tela ou no retrovisor, podendo provocar travadas ou quedas de fps. Com texturas menores a largura de banda interna fica aliviada, podendo carregar e descarregar mais texturas de uma vez.

## Sempre teste no celular, se possível!

Isso é muito importante: tanto os mods de ônibus como de mapas devem ser testados nos celulares antes de lançados, para verificar se tudo está ok. Talvez alguns problemas de performance ou algum bug interno nos colisores gerados pela engine só aconteçam em alguns celulares, mas não no PC... Isso acontece direto com a gente no desenvolvimento: funciona no editor da Unity e no PC do produtor, mas depois apresenta erros doidos nos celulares...

Não há nenhum problema se você quiser fazer um mapa só para PC, nesse caso é bom deixar claro no lançamento de que ele pode não ser adequado aos celulares atuais, evitando que os usuários saiam frustrados.

Mais de 95% dos jogadores do Proton estão nos celulares, vale a pena atingir esse público.

## Isso é tudo por agora!

Muito obrigado pelo interesse em fazer mapas para o Proton Bus!

Se você criar um mapa gratuito bem legal que a galera curtir, não deixe de mostrar nos grupos do jogo!

Boa diversão!