GAME DESIGN

GAME DOCUMENT

Smart City System

Escrito por: Equipe de Desenvolvimento

Data de Exportação: 10/11/2025

Resumo do Jogo

Smart City System

Game Concept

NOME DO JOGO (TÍTULO E SUBTÍTULO DO CONCEITO)

Título do Conceito: SmartCitySystem: Um Gêmeo Digital para Simulação Urbana

GÊNERO

Simulação / Gêmeo Digital (Digital Twin)

PLATAFORMA(S)

- Desktop (PC/Mac/Linux)

SOFTWARES E HARDWARES

Softwares: Unity (C#), Node.js (Express), Python (Flask, scikit-learn, Pandas), MongoDB

Hardwares: Não definido

MODO JOGADOR

Single Player , Multi Player, Cooperativo Local

MEMBROS DA EQUIPE

PRAZOS / CUSTOS

Início Esperado: 09/09/2025

Fim Esperado: 11/11/2025

Game Proposal

QUAL É A FINALIDADE DO JOGO?

1. Replay Histórico: Reconstruir e visualizar o impacto de condições climáticas reais (chuva, temperatura, vento) em um ambiente 3D, usando dados históricos do INMET.

2. Monitoramento em Tempo Real: Coletar dados de sensores de tráfego (fluxo de veículos) e semáforos da simulação.

3. Análise Preditiva: Usar Machine Learning para detectar anomalias (outliers) no tráfego em tempo real e prever a qualidade do ar com base em dados históricos.

QUAL É O OBJETIVO PRINCIPAL?

1. Selecionar uma estação meteorológica, data e hora para iniciar um replay climático histórico.

2. Importar dados de diferentes fontes (INMET, QUALAR) para popular o banco de dados.

3. Treinar os modelos de Machine Learning (Detecção de Outlier e Qualidade do Ar).

4. Navegar pelo ambiente 3D para observar visualmente os impactos (chuva, mudança de céu) e eventos (anomalias de tráfego).

QUAIS AS CONDIÇÕES DE VITÓRIA?

O QUE/QUEM O JOGADOR CONTROLARÁ?

1. Observador (Principal): Controla a interface de usuário (UI) para configurar filtros, datas, importar dados e iniciar a simulação.

2. Agente (Secundário): Pode opcionalmente controlar um veículo em primeira pessoa (PlayerVehicleController.cs) usando o teclado (WASD) para navegar pelo cenário.

QUAL É O CENÁRIO DA HISTÓRIA?

Um ambiente urbano 3D simulado. O cenário contém infraestrutura de tráfego, como cruzamentos, semáforos e sensores de contagem de veículos. O ambiente é dinâmico, respondendo aos dados climáticos com mudanças no céu (usando skyboxDayClear ou skyboxDayOvercast), névoa e efeitos de chuva (usando rainParticleSystem).

QUANTAS HORAS DE JOGO VOCÊ ESPERA?

A duração da simulação é definida pelo usuário.

QUAL O PÚBLICO ALVO?

Planejadores urbanos, pesquisadores acadêmicos, estudantes de engenharia de tráfego ou ciências ambientais, e desenvolvedores interessados em Gêmeos Digitais.

QUAIS OS PRINCIPAIS COMPETIDORES?

O QUE TORNA ESTE JOGO DIFERENTE (INOVAÇÃO)?

1. Uma simulação 3D interativa (Unity).

2. Um backend de API robusto (Node.js) que gerencia dados históricos.

3. Microsserviços de Machine Learning (Python) que realizam predições e detecção de anomalias. O sistema não apenas "repete" o passado (clima), mas também reage a eventos em tempo real (tráfego) e prevê condições futuras (qualidade do ar).

HISTÓRIA DO JOGO / RESUMO DA NARRATIVA

A "história" é o conjunto de dados reais (clima, tráfego, poluição) que o usuário escolhe carregar e observar. O foco é a simulação de dados, não uma trama.

Game Design

Visão Geral do Gameplay

O "gameplay" é centrado na operação de um Gêmeo Digital, focado em configuração, simulação de dados e observação, com elementos de interação em tempo real. O fluxo do usuário é dividido em quatro fases principais:

1. Fase de Preparação e Gerenciamento de Dados
Antes de iniciar a simulação, o usuário atua como um administrador de sistema, preparando o ambiente de dados através da interface de usuário (UI):

Importação de Dados: O usuário utiliza painéis de UI, como o QualarFilterUIManager e o FilterUIManager, para selecionar fontes de dados (ex: INMET, QUALAR). Eles podem disparar a importação de dados históricos para o MongoDB (ex: importar dados de poluição de uma estação específica).

Treinamento de IA: O usuário deve acionar os botões de "Treinar" (ex: trainOutlierButton e o botão de treino do QualarFilterUIManager). Isso executa os scripts Python (train_outlier_model.py, train_air_quality_model.py) no backend. Esses scripts leem os dados do MongoDB, treinam os modelos de Machine Learning (usando LocalOutlierFactor, RandomForestClassifier, SMOTE) e salvam os arquivos .joblib (modelos de ML) no servidor.

2. Fase de Configuração da Simulação
Uma vez que os dados e modelos estão prontos, o usuário configura o cenário de replay:

Filtros em Cascata: O usuário navega por uma série de menus dropdown (TMP_Dropdown). A seleção de um "Ano" habilita o dropdown "Região", que habilita "Estado", que por sua vez habilita "Estação".

Seleção de Tempo: Após escolher uma estação meteorológica válida, o usuário seleciona uma data (Mês, Dia) e uma hora (HH:00) exatas para iniciar o replay histórico.

3. Fase de Execução e Observação (O Gêmeo Digital)
Este é o núcleo do "gameplay", onde o usuário observa a simulação reagir aos dados:

Início da Simulação: O usuário pressiona "Start". O SimulationClock assume o controle, avançando o tempo (ex: 1 hora de simulação a cada gameMinutesPerDataHour minutos reais).

Observação Passiva (Clima): A cada "hora" simulada, o SimulationClock busca os dados climáticos da API de replay. O EnvironmentManager reage a isso:

Se precipitation_total_hourly_mm > 0.1, o rainParticleSystem é ativado e o céu muda para skyboxDayOvercast.

Caso contrário, a chuva para e o céu fica limpo (skyboxDayClear).

Os painéis da UI (S_UIWeatherDisplay) são atualizados em tempo real com os números exatos de temperatura, umidade, vento, etc..

Observação Preditiva (Qualidade do Ar): A cada hora simulada, o SimulationClock também pede uma predição de qualidade do ar. O resultado (ex: "Moderada") é exibido no airQualityPredictionText.

Observação Ativa (Outliers de Tráfego): Durante a simulação, veículos de IA (SimpleCarAI) movem-se pela cidade e passam pelos VehicleCounterSensor.

Reação da IA: A cada intervalo (reportingInterval), o sensor envia os dados de tráfego (contagem de veículos, tempo médio) para a API de ML (/ml/predict/outlier). Se o modelo LOF (Python) detectar uma anomalia e retornar is_outlier_lof == true, o IncidentManager é notificado e instancia automaticamente um incidentPrefab (um carro quebrado, por exemplo) na cena. Este incidente permanece visível por incidentDuration segundos antes de desaparecer.

4. Fase de Interação Direta
Enquanto a simulação automática ocorre, o usuário tem a liberdade de interagir diretamente com o mundo:

Navegação: O usuário pode assumir o controle de um veículo (PlayerVehicleController) e dirigir livremente pelo cenário usando as teclas WASD para inspecionar os eventos (como a chuva, os semáforos ou o incidente de tráfego) de perto.

Sensor de Contagem de Veículos

Controle de Veículos
Pode ter waypoints simples para seguir um caminho.
Pode ter uma lógica básica para parar no semáforo vermelho (verificando o estado do TrafficLightSensor mais próximo).

Variações de Demanda: Crie um sistema (pode ser um script gerenciador na Unity) que simule horários de pico, fluxo normal, ou baixo movimento, alterando a taxa de spawn de veículos e, possivelmente, seus destinos.
Incidentes Aleatórios:
Acidentes/Bloqueios: Simule um bloqueio em uma via (desative um trecho da rota, coloque um obstáculo visual). Seu Gêmeo Digital (os veículos nele) deveria reagir, buscando rotas alternativas. Os sensores virtuais detectariam o congestionamento resultante.
Falha de Semáforo (Virtual): Faça um dos seus semáforos virtuais "falhar" (ex: ficar amarelo piscante ou apagado). Observe como isso impacta o fluxo no Gêmeo Digital.
Eventos Especiais: Simule um evento na cidade (show, jogo) que aumente drasticamente o fluxo para uma região específica.
Condições Climáticas (Simplificadas): Embora seu foco não seja visual, você pode simular o efeito do clima. Ex: em "chuva forte simulada", reduza a velocidade máxima dos veículos virtuais ou aumente a probabilidade de pequenos "incidentes".
Perfis de Comportamento:
Seus veículos virtuais poderiam ter perfis diferentes (alguns mais "apressados", outros mais "cautelosos"), afetando como reagem a semáforos amarelos ou a proximidade de outros veículos.

Simulação de Políticas e Intervenções:
Alterar Tempos de Semáforo: Use seu Gêmeo Digital para testar diferentes programações de semáforos. Altere os tempos no ambiente virtual e observe (através dos seus sensores virtuais e dashboards) o impacto no fluxo, nos tempos de espera, etc. A configuração que gerar melhores resultados no Gêmeo Digital seria uma forte candidata para ser implementada no sistema real.
Novas Rotas ou Vias: "Desenhe" uma nova via ou altere o sentido de uma existente no seu Gêmeo Digital. Simule o tráfego e veja o impacto.
Planejamento de Resposta a Incidentes: Use os cenários de incidentes (item 1) para testar planos de contingência. Se a "Via A" for bloqueada, qual a melhor estratégia de desvio? Teste no Gêmeo Digital.

Impacto nos Comportamentos:
Hora do Dia: Pode influenciar diretamente a densidade do tráfego (horários de pico), a iluminação pública virtual, o comportamento de pedestres virtuais, e até mesmo a eficiência de painéis solares virtuais, se você chegar a esse nível de detalhe.
Clima (Tempo): Chuva, neblina, neve (dependendo da região) podem afetar a velocidade dos veículos, a visibilidade, a aderência (se você simular fisicamente), e também o comportamento de pedestres. Pode até influenciar a demanda por certos serviços.
Dia da Semana/Data: Padrões de tráfego são diferentes em dias úteis, fins de semana e feriados. Eventos específicos podem ser associados a datas.

Excelente! Fico muito feliz em saber que deu tudo certo! □□

Parabéns! Agora você tem uma base funcional no seu projeto SmartCitySystem que busca dados de tempo e clima do mundo real e os exibe na Unity. Isso é um ótimo alicerce para um Gêmeo Digital.

Recapitulando o que alcançamos:

Configuramos a busca de hora atual para um fuso horário específico usando a WorldTimeAPI.
Integramos a busca de condições climáticas atuais (temperatura e descrição) usando a OpenWeatherMap API.
Exibimos essas informações dinamicamente na interface do usuário da Unity.
E agora? Quais os próximos passos para o seu SmartCitySystem?

Com essa base, aqui estão algumas ideias do que você poderia desenvolver em seguida, alinhadas com a sua ideia original de Gêmeos Digitais e simulação urbana:

Interação do Usuário para Seleção de Local:

Em vez de definir a cidade/fuso horário no Inspector, crie campos de input ou dropdowns na UI para que o usuário possa digitar ou selecionar a cidade/país/fuso horário desejado e atualizar os dados dinamicamente.
Usar os Dados Reais para Influenciar a Simulação:

Hora do Dia:
Altere a intensidade do tráfego virtual (quantidade de carros spawnados) com base na hora do dia (simulando horários de pico).
Controle a iluminação pública virtual (acendendo postes à noite).
Clima:
Se o clima for "Rain" (Chuva), você pode reduzir a velocidade máxima dos veículos virtuais ou até mesmo alterar a aparência da cena (ex: um efeito de chuva com partículas, mudar o skybox para um céu nublado).
Reduzir a quantidade de "pedestres virtuais" se o tempo estiver ruim.
Feedback Visual Mais Rico para o Clima:

Além de texto, mude o Skybox da Unity para refletir o tempo (ensolarado, nublado, noite).
Adicione sistemas de partículas para chuva ou neve (se aplicável à região simulada).
Avançar na Simulação dos Sensores Virtuais:

Comece a implementar os "sensores de semáforo inteligente" e "contadores de veículos" que você descreveu no seu paper.
Faça com que os dados gerados por esses sensores virtuais sejam influenciados pela hora e clima (ex: contagem de veículos varia com a hora).
Persistência de Dados (MongoDB):

Comece a enviar os dados gerados (tanto os dados reais buscados quanto os dados dos seus sensores virtuais) para o seu backend Node.js e, em seguida, para o MongoDB, conforme planejado no seu artigo.
Refinamento e Tratamento de Erros:

Melhore o feedback visual para o usuário caso haja falha na busca dos dados das APIs.
Adicione mais robustez ao sistema.

Níveis de Jogo

Nenhum level definido.

Mecânicas do Jogo

Importador Sob-Demanda

Um script separado (ex: build_catalog.js) que você executa uma única vez no seu servidor.

Fase A

Usar a biblioteca (como yauzl ou adm-zip) para ler o conteúdo (os nomes dos arquivos .csv) dentro de cada Zip sem ter que baixar o arquivo inteiro. Se isso não for possível, o script terá que baixar/extrair/catalogar/apagar um por um.

Ele faz o "parse" (análise) de cada nome de arquivo .csv (ex: INMET_SE_SP_A771_SAO PAULO - INTERLAGOS_...csv).

Ele extrai: Regiao: "SE", Estado: "SP", StationID: "A771", StationName: "SAO PAULO - INTERLAGOS", Ano: ....

Ele salva essas informações em uma nova coleção MongoDB chamada inmet_file_catalog.

Fase B

Fase B: API de Filtros Rápida (Backend - Node.js)
O controllers/filterController.js é substituído por uma versão que consulta apenas a nova coleção inmet_file_catalog.

GET /api/filters/years: Retorna os anos únicos do catálogo. (Instantâneo)

GET /api/filters/regions?year=2012: Retorna as regiões únicas do catálogo para aquele ano. (Instantâneo)

GET /api/filters/stations?year=2012&state=SP: Retorna as estações (_id e name) do catálogo para aquele ano e estado. (Instantâneo)

Fase C: Nova UI de Importação (Frontend - Unity)
A UI que criamos (FilterUIManager) agora se torna um "Gerenciador de Importação".

O usuário seleciona Ano, Região, Estado, Estação.

Em vez de um botão "Start Simulation", ele vê um botão "Importar Dados da Estação".

Ao clicar, a Unity chama uma nova API.

Fase D:

Nova API de Importação (Backend - Node.js)
Novo Controller: controllers/importController.js
Nova Rota: POST /api/import/station

Recebe JSON: { "year": 2012, "station_id": "A748" }

Procura no inmet_file_catalog o nome do arquivo (ex: INMET_SE_SP_A748_...csv) e o Zip (ex: 2012.zip).

Baixa o 2012.zip (se ainda não baixou).

Extrai apenas o arquivo CSV específico (INMET_SE_SP_A748...csv) para uma pasta temporária.

Executa a lógica processSingleCsv (que já temos) apenas nesse arquivo.

Salva os dados nas coleções stations e historical_weather.

Apaga o CSV temporário.

Retorna {"success": true, "message": "Estação A748 para 2012 importada com sucesso."}.

Sincronização Ambiental (Replay Climático)

O sistema busca dados meteorológicos históricos (como precipitação e temperatura) do banco de dados com base na hora e data atuais da simulação. Esses dados alteram dinamicamente o ambiente 3D, ativando/desativando efeitos visuais como o sistema de partículas de chuva (rainParticleSystem) e trocando o material do céu (skyboxDayClear / skyboxDayOvercast).

Geração de Incidentes (Detecção de Outlier)

Sensores de tráfego (VehicleCounterSensor.cs) na cena 3D coletam dados em tempo real sobre o fluxo de veículos. Esses dados são enviados para um modelo de Machine Learning (LOF, predict_outlier.py). Se o modelo detectar uma anomalia (um "outlier"), o IncidentManager.cs é notificado e instancia automaticamente um incidentPrefab (um carro quebrado, por exemplo) no local do sensor por um tempo limitado.

Previsão Preditiva (Qualidade do Ar)

Conforme o relógio da simulação avança, o sistema envia o contexto de tempo atual (hora, dia da semana, mês) para um modelo de Machine Learning (RandomForestClassifier). O modelo prevê a classe da qualidade do ar (ex: "Moderada") e seu score de confiança, que são então exibidos em tempo real na interface do usuário (airQualityPredictionText).

Gerenciamento de Dados (Importação e Treinamento)

O usuário interage com painéis de UI (FilterUIManager, QualarFilterUIManager) para disparar processos no backend. Isso inclui a importação de novos dados históricos (ex: QUALAR) para o MongoDB e o acionamento do treinamento dos modelos de Machine Learning, que são necessários para as outras mecânicas funcionarem.

Exploração Livre (Controle de Veículo)

O usuário pode assumir o controle direto de um veículo (PlayerVehicleController.cs) usando o teclado (WASD). Isso permite que ele navegue livremente pelo cenário 3D, inspecione os eventos climáticos ou os incidentes de tráfego gerados pela IA.

Assets do Jogo

Prefab de Incidente (incidentPrefab) (Prefab 3D (Modelo))

Contexto de Uso: IncidentManager.cs

DESCRIÇÃO:

O objeto 3D (ex: um carro quebrado, cones) que é instanciado na cena 3D quando a IA de detecção de outlier (predict_outlier.py) reporta uma anomalia no tráfego.

Modelo de Detecção de Outlier (lof_model.joblib) (Dados (Machine Learning))

Contexto de Uso: train_outlier_model.py, predict_outlier.py, app_outlier.py

DESCRIÇÃO:

O arquivo de modelo LocalOutlierFactor treinado. É o "cérebro" carregado pelo backend Python para analisar os dados de tráfego e decidir se são uma anomalia ou não.

Modelo de Classificação de Ar (air_quality_classifier.joblib) (Dados (Machine Learning))

Contexto de Uso: train_air_quality_model.py, predict_air_quality.py

DESCRIÇÃO:

O arquivo de modelo RandomForestClassifier treinado. É carregado pelo backend Python para prever a qualidade do ar (Boa, Moderada, Alerta) com base na hora e dia.

Efeito de Chuva (rainParticleSystem) (Efeito Visual (VFX))

Contexto de Uso: EnvironmentManager.cs

DESCRIÇÃO:

Um Sistema de Partículas do Unity que é ativado (Play()) ou desativado (Stop()) pelo EnvironmentManager para simular chuva, com base nos dados de precipitação recebidos do ReplayManager.

Skybox (Claro / Nublado) (Material (Ambiente))

Contexto de Uso: EnvironmentManager.cs (referencia skyboxDayClear, skyboxDayOvercast)

DESCRIÇÃO:

Materiais de céu (skybox) que são trocados dinamicamente para refletir a condição climática atual da simulação (ex: skyboxDayOvercast quando está chovendo).

Materiais do Semáforo (Ligado/Desligado) (Material (3D))

Contexto de Uso: TrafficLightSensor.cs (referencia redOnMaterial, greenOnMaterial, yellowOnMaterial, lightOffMaterial)

DESCRIÇÃO:

Materiais que são aplicados aos renderers 3D do semáforo para mudar visualmente seu estado (Vermelho, Amarelo, Verde, Desligado) conforme o ciclo avança.

Prefab de Veículo de IA (Prefab 3D (Modelo))

Contexto de Uso: SimpleCarAI.cs, VehicleCounterSensor.cs

DESCRIÇÃO:

O modelo 3D usado para os carros de IA. Deve ser anexado ao script SimpleCarAI.cs e ter a tag "Vehicle" para ser detectado pelo VehicleCounterSensor.

Prefab do Veículo do Jogador (Prefab 3D (Modelo))

Contexto de Uso: PlayerVehicleController.cs

DESCRIÇÃO:

O modelo 3D que o usuário controla diretamente usando o teclado (WASD/Setas), conforme definido no PlayerVehicleController.cs.

Elementos de Interface (UI)

Painel de Filtros (INMET / Replay)

DESCRIÇÃO:

O principal painel de configuração da simulação, referenciado em FilterUIManager.cs. Contém uma série de dropdowns (TMP_Dropdown) em cascata (Ano -> Região -> Estado -> Estação) e seletores de data/hora (Mês, Dia, Hora) que permitem ao usuário definir o ponto de partida exato para o replay histórico.

Painel de Gerenciamento (Qualidade do Ar)

DESCRIÇÃO:

Uma interface dedicada (QualarFilterUIManager.cs) para o Pipeline 2 de ML. Permite ao usuário selecionar Ano, Estação e Parâmetro dos dados QUALAR, disparar a importação desses dados para o MongoDB e acionar o treinamento (trainButton) do modelo de IA de qualidade do ar.

Painel de Controle da Simulação

DESCRIÇÃO:

O conjunto de botões (startButton, stopButton) referenciado em SimulationClock.cs. Permite ao usuário iniciar e parar o avanço do tempo da simulação (o replay histórico).

Display de Status da Simulação

DESCRIÇÃO:

Um conjunto de campos de texto (TextMeshProUGUI) que fornecem feedback em tempo real sobre o estado da simulação, incluindo:

simulationTimeText: Mostra a data e hora (UTC) atual da simulação.

airQualityPredictionText: Mostra a predição da IA (ex: "Qualidade do Ar: Moderada (90%)").

statusText: Um campo genérico (FilterUIManager.cs) para feedback (ex: "Carregando Estações...", "Treinamento concluído!").

Painel de Clima Detalhado

DESCRIÇÃO:

Um painel de texto (detailedWeatherDataText em SimulationClock.cs) que exibe a lista completa de todos os dados meteorológicos recebidos da API de replay para a hora atual da simulação (Temperatura, Precipitação, Vento, Pressão, etc.).

Painel de Clima (Mundo Real)

DESCRIÇÃO:

Um painel de UI opcional (RealWorldDataManager.cs) que exibe a hora e o clima do mundo real (obtidos da WorldTimeAPI e OpenWeatherMap), usado para os sensores em tempo real, como o VehicleCounterSensor.

Painel de Clima Modular

DESCRIÇÃO:

Um painel de UI (definido em S_UIWeatherDisplay.cs) que exibe dados climáticos de forma modular. O usuário pode mapear campos de texto individuais (ex: "Temperatura") para dados específicos (ex: WeatherDataType.Temperature), permitindo painéis personalizados.

Botões de Gerenciamento de IA

DESCRIÇÃO:

Botões (trainOutlierButton, trainButton) que permitem ao usuário disparar os processos de treinamento dos modelos de Machine Learning no backend (ex: RequestOutlierTraining em APIManager.cs).

Animações do Jogo

Geração de Chuva (Início/Fim)

Evento de Disparo: O ReplayManager dispara o evento OnWeatherDataUpdated. O EnvironmentManager.cs recebe os dados e verifica se precipitation_total_hourly_mm > 0.1.

DESCRIÇÃO:

O rainParticleSystem (um Sistema de Partículas do Unity) é ativado (Play()) para começar a chover na cena, ou desativado (Stop()) para parar a chuva.

Mudança de Céu (Clima)

Evento de Disparo: O ReplayManager dispara o evento OnWeatherDataUpdated. O EnvironmentManager.cs recebe os dados de precipitação.

DESCRIÇÃO:

O material do Skybox da cena é trocado dinamicamente. Se houver chuva, o céu muda para skyboxDayOvercast. Se não houver chuva, muda para skyboxDayClear.

Geração de Incidente (Anomalia de Tráfego)

Evento de Disparo: O IncidentManager.cs recebe um callback de sucesso da APIManager.Instance.RequestOutlierCheck. A função HandleOutlierResponse faz o parse do JSON e confirma que analysis.is_outlier_lof == true.

DESCRIÇÃO:

O IncidentManager instancia (Instantiate()) o incidentPrefab (ex: um carro quebrado) na posição do sensor que detectou a anomalia. O objeto é destruído automaticamente após incidentDuration segundos.

Mudança de Sinal do Semáforo

Evento de Disparo: O coroutine LightCycle() no script TrafficLightSensor.cs completa seu tempo de espera (ex: greenDuration, redDuration).

DESCRIÇÃO:

O script troca os materiais (redOnMaterial, yellowOnMaterial, greenOnMaterial, lightOffMaterial) nos renderers 3D do semáforo para mudar visualmente a cor da luz acesa.

Movimento do Veículo (IA)

Evento de Disparo: Update() (contínuo, a cada frame) no script SimpleCarAI.cs.

DESCRIÇÃO:

O script aplica transform.Translate (para mover para frente) e transform.Rotate (para girar em direção ao próximo waypoint), criando a animação de movimento contínuo do tráfego.

Movimento do Veículo (Jogador)

Evento de Disparo: Update() (contínuo, a cada frame) no script PlayerVehicleController.cs.

DESCRIÇÃO:

O script verifica a entrada do teclado (WASD/Setas) e aplica transform.Translate e transform.Rotate para mover e girar o veículo controlado pelo jogador.

Sons do Jogo

Chuva (Loop)

Contexto de Uso: Disparado pelo EnvironmentManager.cs quando o rainParticleSystem é ativado (Play()).

Volume Sugerido: 40%

DESCRIÇÃO:

Um som ambiente de chuva (looping) que toca enquanto os dados climáticos (precipitation_total_hourly_mm) indicarem chuva.

Incidente Detectado (Alerta)

Contexto de Uso: Disparado pelo IncidentManager.cs no momento exato em que ele instancia o incidentPrefab (após a IA detectar um outlier).

Volume Sugerido: 80%

DESCRIÇÃO:

Um som de alerta curto e chamativo (ex: "bip-bip-bip" de um radar ou um alerta de "anomalia detectada") para notificar o usuário que a IA gerou um incidente.

Motor do Veículo (IA)

Contexto de Uso: Toca continuamente enquanto o SimpleCarAI.cs está em movimento (Update()).

Volume Sugerido: 30%

DESCRIÇÃO:

Um som de motor de carro genérico em loop. O volume deve ser baixo e posicionado espacialmente (Áudio 3D) no prefab do veículo.

Motor do Veículo (Jogador)

Contexto de Uso: Toca continuamente enquanto o PlayerVehicleController.cs está em movimento (quando o usuário pressiona WASD).

Volume Sugerido: 60%

DESCRIÇÃO:

Um som de motor de carro em loop, que idealmente mudaria o pitch com a velocidade (embora o script atual não tenha essa lógica).

Clique de Botão (UI)

Contexto de Uso: Disparado em todos os Button.onClick.AddListener() (ex: startButton em SimulationClock.cs, importButton em FilterUIManager.cs).

Volume Sugerido: 50%

DESCRIÇÃO:

Um som de clique de interface padrão para dar feedback tátil ao usuário.

Início de Simulação (UI)

Contexto de Uso: Disparado pelo SimulationClock.cs na função StartSimulation().

Volume Sugerido: 70%

DESCRIÇÃO:

Um som positivo de "ativação" ou "confirmação" para indicar que a simulação complexa foi iniciada com sucesso.

Erro (UI)

Contexto de Uso: Disparado nos callbacks de erro do APIManager.cs (ex: onError?.Invoke(...) em RequestImport ou RequestOutlierTraining).

Volume Sugerido: 75%

DESCRIÇÃO:

Um som de "erro" ou "aviso" (ex: "buzz" ou bip negativo) para alertar o usuário quando uma chamada de API falha (ex: falha na importação ou treinamento).

Esquema de Controles

Interagir com a UI / Clicar em Botão

Tecla/Botão: Botão Esquerdo do Mouse **Input:** Clique

DESCRIÇÃO/NOTAS:

Usado para selecionar opções nos dropdowns (TMP_Dropdown) e clicar em botões como "Start", "Stop", "Importar" e "Treinar". É o método de entrada principal para configurar a simulação.

Veículo

Tecla/Botão: W / Seta para Cima, S / Seta para Baixo, A / Seta para Esquerda, D / Seta para Direita **Input:** Manter Pressionado (Hold)

DESCRIÇÃO/NOTAS:

Move o veículo do jogador para frente.
Move o veículo do jogador para trás.
Gira o veículo do jogador para a esquerda.
Gira o veículo do jogador para a direita.
(Definido em PlayerVehicleController.cs).

Personagens

O Operador (Usuário)

PAPEL:

Player

HISTÓRIA DE FUNDO / BIOGRAFIA:

É o usuário final (planejador urbano, pesquisador ou estudante) que utiliza o Gêmeo Digital para configurar cenários, executar simulações e analisar os resultados visuais e de dados.

HABILIDADES / SKILLS PRINCIPAIS:

Controle de UI: Capaz de operar todos os painéis de filtro (ex: FilterUIManager.cs, QualarFilterUIManager.cs).

Gerenciamento de Dados: Pode disparar a importação de dados e o treinamento de modelos de IA.

Controle de Simulação: Inicia e para o tempo da simulação (SimulationClock.cs).

Pilotagem: Pode assumir o controle de um veículo para exploração (usando PlayerVehicleController.cs).

OUTRAS NOTAS RELEVANTES:

É o único agente humano no sistema.

Veículo de IA

PAPEL:

NPC

DESCRIÇÃO DA APARÊNCIA:

Qualquer prefab de veículo 3D ao qual o script SimpleCarAI.cs é anexado.

PERSONALIDADE:

Previsível, Constante.

HISTÓRIA DE FUNDO / BIOGRAFIA:

Uma entidade simulada cuja única função é navegar pelo cenário 3D seguindo uma rota pré-definida de waypoints para simular o fluxo de tráfego urbano.

HABILIDADES / SKILLS PRINCIPAIS:

Navegação por Waypoint: Segue uma lista de waypoints em loop.

Geração de Eventos: Aciona os triggers dos VehicleCounterSensor.cs ao entrar e sair de suas zonas de detecção.

OUTRAS NOTAS RELEVANTES:

Este agente é essencial para alimentar o modelo de Detecção de Outlier em tempo real.

Gerenciador de Incidentes

PAPEL:

NPC

DESCRIÇÃO DA APARÊNCIA:

Não possui aparência física. Sua manifestação é o incidentPrefab (ex: um carro quebrado, cones) que ele instancia na cena 3D.

PERSONALIDADE:

Reativo, Alerta.

HISTÓRIA DE FUNDO / BIOGRAFIA:

Um sistema de vigilância invisível (IncidentManager.cs) que monitora constantemente o feedback da API de Machine Learning.

HABILIDADES / SKILLS PRINCIPAIS:

Interpretação de IA: Recebe e analisa a resposta JSON da API de outlier (HandleOutlierResponse).

Instanciação de Evento: Se a API de ML (predict_outlier.py) confirmar um outlier (is_outlier_lof == true), este agente tem a habilidade de instanciar um incidentPrefab na posição do sensor.

Limpeza Automática: Remove o incidente da cena após um tempo (incidentDuration).

OUTRAS NOTAS RELEVANTES:

É a principal conexão visual entre a detecção de dados abstrata (Python) e o mundo 3D interativo (Unity).

Itens do Jogo

Dados Climáticos Históricos (INMET) (Tipo: Recurso/Moeda) - Raridade: Comum (Necessário para a simulação base)

DESCRIÇÃO:

Registros de dados meteorológicos reais, importados do INMET e armazenados no MongoDB.

EFEITOS/ATRIBUTOS:

Contém valores (float) para precipitation_total_hourly_mm, temperature_air_dry_bulb_c, wind_speed_hourly_m_s, etc.

NOTAS ADICIONAIS:

Usado pelo EnvironmentManager e S_UIWeatherDisplay para recriar o clima histórico na simulação.

Dados de Qualidade do Ar (QUALAR) (Tipo: Recurso/Moeda) - Raridade: Comum (Necessário para o Pipeline 2 de ML)

DESCRIÇÃO:

Registros históricos de poluição importados para a coleção air_quality_data no MongoDB.

EFEITOS/ATRIBUTOS:

Contém valores (float) para poluentes (ex: mp10, parameter) e um timestamp.

NOTAS ADICIONAIS:

Usado pelo script train_air_quality_model.py para treinar o classificador de IA.

Dados de Fluxo de Tráfego (Sensor) (Tipo: Recurso/Moeda) - Raridade: Comum (Gerado em tempo real)

DESCRIÇÃO:

Coletados em tempo real pelos VehicleCounterSensor e salvos na coleção sensor_vehicle_flow do MongoDB.

EFEITOS/ATRIBUTOS:

Contém vehicleCount (int), avgTimeOnSensor (float), e hora_do_dia (int).

NOTAS ADICIONAIS:

Usado tanto para treinar (train_outlier_model.py) quanto para alimentar (predict_outlier.py) o modelo de Detecção de Outlier.

Modelo de Detecção de Outlier (lof_model.joblib) (Tipo: Item Chave) - Raridade: Único (Gerado pelo usuário)

DESCRIÇÃO:

Um arquivo (.joblib) contendo um modelo LocalOutlierFactor treinado.

EFEITOS/ATRIBUTOS:

predict(X): Retorna -1 para outlier, 1 para inlier. score_samples(X): Retorna o grau de anormalidade.

NOTAS ADICIONAIS:

Este é o "cérebro" da detecção de incidentes. É carregado pelo predict_outlier.py.

Modelo de Classificação de Ar (air_quality_classifier.joblib) (Tipo: Item Chave) - Raridade: Único

(Gerado pelo usuário)

DESCRIÇÃO:

Um arquivo (.joblib) contendo um RandomForestClassifier treinado.

EFEITOS/ATRIBUTOS:

Retorna a classe (ex: "Boa", "Moderada", "Alerta"). predict_proba(X): Retorna o score de confiança.

NOTAS ADICIONAIS:

Usado pelo predict_air_quality.py para alimentar a UI do SimulationClock.

Prefab de Incidente (incidentPrefab) (Tipo: Item Chave) - Raridade: Raro (Aparece apenas em detecção de outlier)

DESCRIÇÃO:

O objeto 3D que é instanciado pelo IncidentManager quando a IA detecta um outlier de tráfego.

EFEITOS/ATRIBUTOS:

Instantiate(): Cria uma representação visual (ex: carro quebrado) na cena. Destroy(): Desaparece após incidentDuration segundos.

NOTAS ADICIONAIS:

É a principal recompensa visual da mecânica de Detecção de Anomalia.

Veículo de IA (Tipo: Outro) - Raridade: Comum

DESCRIÇÃO:

Uma entidade de simulação (SimpleCarAI.cs) que segue waypoints para gerar dados de tráfego.

EFEITOS/ATRIBUTOS:

moveSpeed, turnSpeed. Segue um array de waypoints.

NOTAS ADICIONAIS:

Aciona os triggers dos VehicleCounterSensor para criar os "Dados de Fluxo de Tráfego".

Veículo Controlável (Jogador) (Tipo: Outro) - Raridade: Único

DESCRIÇÃO:

A entidade (PlayerVehicleController.cs) que o usuário pode pilotar (WASD) para explorar o cenário.

EFEITOS/ATRIBUTOS:

moveSpeed, turnSpeed. Controlado por WASD/Setas.

NOTAS ADICIONAIS:

É o "avatar" do usuário no mundo 3D.

Efeito Visual de Chuva (rainParticleSystem) (Tipo: Outro) - Raridade: Comum (Dependente do clima)

DESCRIÇÃO:

Um sistema de partículas ativado pelo EnvironmentManager quando os dados climáticos (precipitation_total_hourly_mm) indicam chuva.

EFEITOS/ATRIBUTOS:

Play(): Inicia a emissão de partículas de chuva. Stop(): Para a emissão.

NOTAS ADICIONAIS:

É a principal manifestação visual da mecânica de Replay Climático.

Skybox (Clima) (Tipo: Outro) - Raridade: Comum

DESCRIÇÃO:

Materiais de céu (skyboxDayClear, skyboxDayOvercast) que são trocados pelo EnvironmentManager para refletir as condições climáticas (chuva ou sol).

EFEITOS/ATRIBUTOS:

Define a aparência visual do céu (textura).

NOTAS ADICIONAIS:

Trabalha em conjunto com o rainParticleSystem.

Sensor de Tráfego (VehicleCounterSensor) (Tipo: Outro) - Raridade: Comum

DESCRIÇÃO:

Um objeto trigger na cena (VehicleCounterSensor.cs) que detecta os veículos de IA, calcula o tempo médio e a contagem.

EFEITOS/ATRIBUTOS:

OnTriggerEnter, OnTriggerExit. Detecta GameObjects com a tag "Vehicle".

NOTAS ADICIONAIS:

É a fonte de dados primária para o Pipeline 1 de ML.

Semáforo (TrafficLightSensor) (Tipo: Outro) - Raridade: Comum

DESCRIÇÃO:

Uma entidade de simulação (TrafficLightSensor.cs) que muda de cor e envia seu estado para o banco de dados.

EFEITOS/ATRIBUTOS:

Muda de estado (Green, Yellow, Red) em intervalos de tempo fixos (greenDuration, yellowDuration, redDuration).

NOTAS ADICIONAIS:

Muda a aparência visual usando redOnMaterial, greenOnMaterial, etc.