Mob Town

Felipe Buchabqui Saenger Marzanasco[1]* Fernando Pinho Marson[1]

[1]Universidade do Vale do Rio dos Sinos, Jogos Digitais, Brasil

Figura 1: Logo do jogo MobTown.

RESUMO

O presente artigo busca mostrar o processo de desenvolvimento de um jogo digital de gênero simulação, com elementos de *tycoon*. Baseado em uma análise de mercado, o trabalho mostra componentes que diferem Mob Town dos jogos do seu gênero, como por exemplo o uso do clima do usuário como entrada de jogo. As etapas de criação são comentadas e no final analisadas com base em testes, que detalham os principais erros e acertos do projeto.

Palavras-chave: Jogos Digitais; Mobilidade Urbana, Clima Real, Tycoon, Simulação;

1 INTRODUÇÃO

O mercado de jogos digitais pode ser dividido em três grandes áreas: Consoles, Computadores e Aparelhos Móveis. Em 2019, a plataforma de PC´s (*Personal Computer*) foi a segunda maior contribuinte para a indústria, gerando aproximadamente 25% dos lucros totais, de acordo com o site Tecmundo[18]. No mesmo ano, os dispositivos móveis geraram uma renda próxima de 54% , sendo então considerados os maiores contribuidores do mercado. Com base nesses dados, o presente artigo busca mostrar o processo de desenvolvimento do jogo Mob Town e uma análise do mercado no qual ele está inserido. O jogo em questão possui gênero de simulação com elementos de *tycoon*.

1.1 Objetivo Geral

Este artigo, feito para fins de projeto final do curso de Jogos Digitais da Escola Politécnica Unisinos, possui como objetivo principal relatar tecnicamente o desenvolvimento de Mob Town. No jogo é possível ser o proprietário e administrador de uma companhia de táxi, bem como motorista. Nesse contexto, o jogo explora mecânicas comuns de simulação de veículos, como na grande franquia Grand Theft Auto, criado pela Rockstar Games[10]. Também possui um modelo de jogabilidade similar ao de jogos de estilo *tycoon*, como em Roblox, desenvolvido pela Roblox Corporation[6].

*e-mail: lipemarzanasco@gmail.com

1.2 Objetivos Específicos

Analisar títulos importantes dos gêneros simulação e *tycoon* no mercado, mostrando suas características, avaliações e como elas influenciaram o desenvolvimento do projeto é um dos objetivos secundários. Além disso, comentar sobre a experiência de realizar uma integração com o mundo externo do jogador, utilizando API's como a Open Weather Map[15].

1.3 Estrutura do Artigo

O artigo está dividido em sete seções, sendo que a primeira é a introdução. A segunda é uma análise de mercado que explicita inspirações, concorrentes e monetização do segmento. Na terceira etapa, comenta-se sobre o jogo, com subseções de *gameplay*, elementos de *gameplay* e mecânicas. A quarta parte desmembra e elucida o processo de desenvolvimento do jogo. A quinta apresenta os testes de *gameplay*, que foram realizados a partir de um link disponibilizado online, enquanto a sexta discorre os resultados da seção anterior. Já a última etapa aponta as considerações finais, com ideias de aprimoramentos na subseção trabalhos futuros e menções de agradecimento.

2 ANÁLISE DE MERCADO

Como comentado na Introdução, o mercado de jogos digitais pode ser dividido em três grandes áreas: Consoles, Computadores e Dispositivos Móveis, segundo o artigo de Willians[28]. Segundo o site Tecmundo[18], no ano de 2019 a indústria aqui comentada gerou aproximadamente 120 bilhões de dólares norte-americanos, o que no mesmo período chegava a aproximadamente 465 bilhões de reais (visto que, na época, um dólar estava valendo R$3,88 de acordo com a Morningstar[16]). Deste total arrecadado, 25% vem do segmento de PC´s, 54% de dispositivos móveis e 21% dos consoles[18].

Um dos progenitores dos jogos de simulação econômica é o The Sumerian Game lançado no ano de 1964 e desenvolvido pela empresa norte-americana IBM. Este jogo era baseado em textos, sem visual e era centrado no gerenciamento de recursos para o funcionamento de uma cidade. As escolhas do jogador afetavam a maneira como o *mainframe*[1] resolvia as simulações. Desta forma, o ambiente era responsivo as atitudes do *player*, fazendo com que cada jo-

[1]estes são computadores de grande porte, com uma incrível capacidade

gador obtivesse uma maneira diferente de evoluir o seu município. Exemplo de referência à Figura 4.

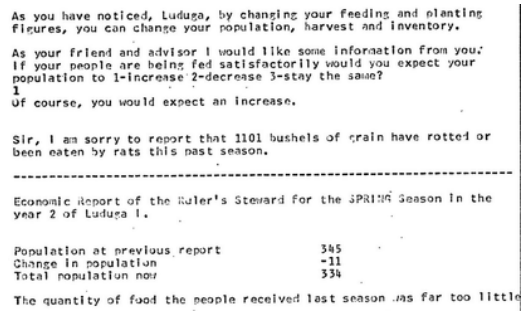

Figura 2: The Sumerian Game - Gameplay.

No ano de 1983 foi lançado o jogo M.U.L.E., desenvolvido pela Blue Planet Software & Ozark Softscape. Este jogo é uma simulação de oferta e demanda, onde até quatro jogadores competem entre si para acumular riquezas e também cooperam para a sobrevivência de sua colônia.

Em 1989, chegou para o mercado o SimCity, criado por Will Wrigth. No jogo deve-se construir e administrar uma cidade, porém de uma forma mais realista, com residências, áreas industriais e comerciais e questões referentes a sociedade política e civil. Desta forma, é proposto ao usuário que sejam elaboradas estratégias para lidar com as simulações. Este jogo premiado marcou o gênero de simulação, pois obteve análises positivas de críticos que, na época, não estavam acostumados com esse estilo de *gameplay*.

Figura 3: SimCity 1989 - Game.

Esses jogos, assim como outros, colaboraram para o desenvolvimento do que conhecemos hoje como o gênero *tycoon* e simulação. Atualmente é possível citar, por exemplo, Zoo Tycoon, publicado em 2013, desenvolvido pelas empresas Frontier Develpments & Blue Fang Games. Neste jogo, o usuário deve cuidar e desenvolver o seu zoológico, pensando em diversas questões como bem estar dos animais, infraestrutura para visitantes, economia da empresa, etc. Hoje, possui uma avaliação positiva na loja de softwares Steam[22], com 61% das análises consideradas boas.

É valido destacar, também, jogos que utilizam a localização do usuário em tempo real, os chamados *Geolocation Based Game*, onde uma das entradas é a localização do jogador. Fazem uso desta mecânica jogos como Pokemon Go, desenvolvido pela empresa norte-americana Niantic, e Minecraft Earth, produzido pela Mojang Studios. Por mais que estes não sejam jogos de gestão de recursos e nem façam parte da plataforma de computadores, possuem o mapa e o clima do jogo dependentes da localização do usuário. Esta ferramenta foi um diferencial que auxiliou ambos a obterem sucesso e uma grande quantidade de downloads. Pokemon Go, por exemplo, no ano de 2018 alcançou a marca de 800 milhões de downloads e, Minecraft Earth, em 2019, 2.1 milhões.

Mob Town procura unir as mecânicas dos jogos mencionados com elementos de *tycoon* e simulação visto em projetos como Zoo Tycoon e SimCity. Além de possibilitar uma imersão pelo uso da geolocalização como *input*, assim como em Pokemon Go.

2.1 Inspirações

Jogos mencionados, como Zoo Tycoon, Pokemon Go, Grand Theft Auto, Sim City, Roblox e Simutrans são grandes inspirações para o projeto Mob Town. Pokemon Go usufrui da localização do usuário como entrada de dados no jogo, onde o clima da cidade real do jogador interfere no ambiente do jogo. Zoo Tycoon, por sua vez, é um exemplo atual do estilo de jogos tycoon, pois traz a simulação da criação de uma 'empresa', que, no caso, atua na área de zoológicos.

Grand Theft Auto possui grande influência visual no projeto, por mais que não tenha a ver com o gênero de Mob Town. Também é importante ressaltar o modelo de City Tycoon do Roblox e de SimCity - 2014, uma vez que alguns elementos de simulação de cidades são compatíveis com os do projeto. O jogo Simutrans, produzido pela empresa Simutrans Development, serve como fonte de ideias para o projeto, visto que, assim como Mob Town, este *tycoon* possui temática de mobilidade urbana.

As inspirações aqui mencionadas ajudaram também no quesito de definição da plataforma alvo, pois, todos os jogos apresentados (exceto Pokemon Go) são para PC´s. Mesmo os que já possuem versões mobile começaram nos computadores.

2.2 Concorrentes

Os concorrentes de Mob Town são jogos de estilo *tycoon*, com gerenciamento de empresas e voltados a simulação de ambientes urbanos, como Roblox, Simutrans e SimCity-2014. Roblox possui uma avaliação quatro estrelas na loja de jogos da Microsoft e já possui sua versão para dispositivos móveis, avaliado com uma nota de 4,5 de 5 na loja para aparelhos iOS, AppStore[3].

Simutrans, por sua vez, consta com análises "ligeiramente positivas"na loja de jogos para computadores Steam. SimCity aparece com críticas "ligeiramente positivas"na Steam, com 5.137 mil avaliações. A sua versão mobile, na loja AppStore, dispõe de 4 estrelas e mais de 4.3 mil classificações, sendo considerada, na mesma loja, a nona colocada no ranking de melhores *app´s* do gênero.

De forma geral, os jogos aqui mencionados possuem boa avaliação e destaque no seu meio. Mob Town busca se diferenciar destas referências trazendo uma jogabilidade mais dinâmica, interativa e imersiva, contando com elementos vistos em jogos como GTA, somados à geolocalização e, ao mesmo tempo, atento a simulação econômica.

2.3 Monetização

Existem algumas maneiras de se pensar em monetização em um contexto de jogos digitais de acordo com o TEDJE (Time de Ensino de Desenvolvimento de Jogos Eletrônicos)[5]. Existem as chamadas vendas únicas, onde o jogador paga somente uma vez pelo produto, e mensalidades, onde a fim de continuar jogando, o usuário deve pagar uma quantia previamente definida por mês. Há a possibilidade de deixar o jogo gratuito, porém com anúncios e/ou microtransações. Cabe ao desenvolvedor do projeto avaliar e estudar seu jogo e o mercado que o mesmo está incluído, para então realizar esta escolha.

Com base nessa informação, esta subsecção mostrará como alguns jogos anteriormente apresentados realizam o processo de monetização e como isso é pensado para Mob Town.

A renda de Roblox é obtida através do uso de assinaturas, onde os jogadores que querem algumas vantagens pagam um valor X de

de processamento. Na época de The Sumerian Game, eram utilizados para simulações estudantis e pesquisas.

acordo com o tipo de assinatura desejada. No jogo existem três tipos de assinaturas: Roblox Premium 450, custando aproximadamente R$27; Roblox Premium 1000, custando cerca de 55 reais; Roblox Premium 2200, valendo R$110. Além disso, o jogo conta com a compra de cosméticos, isto é, itens dentro do jogo, sejam eles somente visuais ou não.

Zoo Tycoon, por sua vez, obtém dinheiro utilizando o método de compra única, ou seja, seus jogadores adquirem o jogo completo pagando somente uma única vez. O jogo esta disponível para download por 37,99 reais na Steam[22].

Com base nessas informações, Mob Town pretende adotar a forma gratuita de jogar, entretanto, contendo microtransações feitas por meio da aquisição de cosméticos *in-game*. Se futuramente portado para plataforma mobile, o projeto continuará sendo gratuito. No entanto, contará com mais uma forma de renda, os anúncios.

3 Jogo

Mob Town é um jogo digital de simulação com elementos de estratégia, onde o jogador deve montar seu 'império' de mobilidade urbana em uma cidade fictícia. Seu objetivo é obter lucro e, com isso, deixar sua vida mais fácil, comprando as vantagens disponibilizadas durante a *gameplay*. Para conseguir o almejado é necessário atender aos chamados dos clientes, realizando as ações estipuladas com cuidado e, estas, quando concluídas, retornam, ou não, lucro para o jogador. O jogo em questão é 3D, câmera em terceira pessoa e foi desenvolvido para a plataforma de computadores.

O clima do jogo é o mesmo clima da cidade real escolhida pelo jogador e, por isso, interfere na funcionalidade de algumas mecânicas. Se, por exemplo, a temperatura local estiver alta e o jogador ainda não tiver adquirido ar condicionado, ele terá um retorno financeiro menor. A ideia de Mob Town foi concebida na cadeira de Projeto Final II. O autor do projeto optou em reformular a ideia antecessora desenvolvida na disciplina de Projeto Final I, uma vez que a mesma possuía elementos de simulação muito criteriosos, sendo assim, mais difíceis de desenvolver no prazo estipulado.

3.1 *Gameplay*

A *gameplay* central do projeto consiste em o jogador simular o dia a dia de um dono de uma empresa de táxi. Para isso, é necessário pegar passageiros e levá-los para seus respectivos endereços. Ao realizar esta ação, o jogador é recompensado com dinheiro, o que o possibilita fazer *upgrades* e facilita sua jornada.

O jogador, representado dentro do jogo com um táxi, deve evitar ao máximo colisões, uma vez que estas causam danos ao veículo, dificultando a direção e podendo causar mal estar aos passageiros, que podem se frustrar com a viagem. Problemas no desempenho do motorista levam incertezas para o restante da população quanto a qualidade da empresa liderada pelo *player*.

3.2 Elementos de *Gameplay*

Esta subseção contempla os elementos que constituem a *gameplay* do jogo.

3.2.1 Jogador

O jogador de Mob Town possui a função de fazer a empresa funcionar adequadamente. É de sua responsabilidade controlar o táxi, pegar passageiros e utilizar o celular. Ao controlar o táxi, o jogador possui a habilidade de se mover pela cidade, podendo então pegar passageiros que estejam a sua espera.

3.2.2 Pedestres

Os pedestres fazem parte da simulação de um ambiente urbano, sua função é dar vida à cidade, fazendo com que a simulação do mundo do jogo fique mais imersiva. Os pedestres possuem a liberdade de andar pela cidade, no entanto, existem algumas regras que precisam seguir, como andar somente nas calçadas, atravessar a rua na faixa de pedestres e seguir até seu ponto de destino. Uma vez que a condição de chegada ao destino seja confirmada, é feita a escolha de um novo ponto meta, fazendo com que o pedestre volte ao seu ciclo. Mais detalhes do desenvolvimento serão abordados na Seção 4.

3.2.3 Veículos

Assim como os pedestres, o objetivo dos veículos também é aumentar a imersão e realçar a simulação do ambiente tratado no jogo. No entanto, eles possuem mais uma função: ser um obstáculo para o jogador. Cada automóvel possui seus próprios parâmetros e destino, a fim de causar um comportamento diferente entre os meios de transporte da cidade.

3.2.4 Mecânico

O mecânico é a loja de *hardware* de Mob Town. Além de possuir a função de reparar o carro do jogador, o mesmo é responsável pela realização de melhorias físicas no táxi operado pelo usuário.

Cada *upgrade* possui sua precificação e o valor do reparo do veículo varia de acordo com a quantidade de dano sofrido. As melhorias disponíveis no momento são: *Air Conditionair*, *Increase Resistance* e *Increase Confort*.

- **Air Conditionair**: Responsável por deixar mais amena a temperatura do carro, fazendo com que os passageiros fiquem mais confortáveis, consequentemente melhorando a avaliação da viagem.
- **Increase Resistance**: Responsável por deixar o táxi mais resistente a danos. Melhora em 5% a resistência do veículo a cada compra.
- **Increase Confort**: Responsável por deixar o táxi mais confortável. Melhora em 5% o conforto do veículo a cada compra.

3.2.5 Celular

O jogador pode utilizar o celular para verificar diversas informações sobre temperatura, horário, condição financeira, mapa da cidade e local de passageiros. Para isso o aparelho conta com aplicativos, estes são: Mob Mail, Mob Store, Mob Map e Mob Bank.

- **Mob Mail:** Possui a função de informar a localização dos passageiros ao jogador através de mensagens de texto.
- **Mob Store:** Esta aplicação possui a finalidade de ser a loja de softwares. Aqui é onde o jogador pode fazer aquisições de outros programas.
- **Mob Map:** Este app mostra ao usuário o mapa da cidade, contendo sua localização e o nome de cada rua.
- **Mob Bank:** Informa ao jogador a sua situação financeira.

3.2.6 Cidade

O mapa de Mob Town é um item importante na jogabilidade, pois o mesmo foi feito com o objetivo de auxiliar o jogador. Todas as ruas da cidade são interligadas e possuem sentido duplo, ou seja, existem diversos caminhos que o jogador pode fazer para chegar ao seu destino. Todas as ruas possuem placas com nome e informações sobre a rua seguinte.

3.2.7 Dia e Noite

O jogo possui sistema próprio de horário, possuindo seus ciclos de dia e noite. O objetivo desta funcionalidade é realçar a simulação do ambiente, aumentando a imersão dentro do jogo.

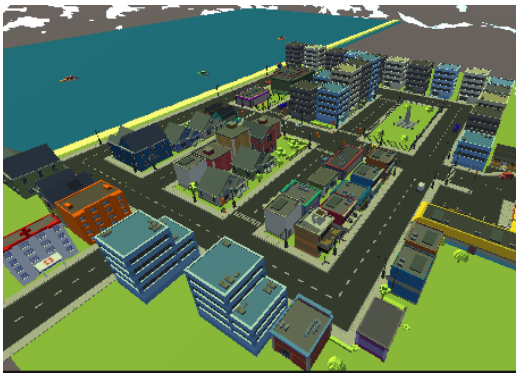

Figura 4: Cidade Mob Town - Fonte: Autor.

3.3 Mecânicas

No projeto Mob Town existem as seguintes mecânicas: Movimentação do Carro, Busca de Passageiros, Avaliação da Viagem, Condição do Veículo, Reparos, Upgrades de *hardware* e *software* e Clima Vigente. A seguir são explicadas as funcionalidades de cada uma.

3.3.1 Movimentação do Carro

Esta *feature* consiste em o usuário poder controlar o automóvel do jogo, habilitando a locomoção do jogador pela cidade.

3.3.2 Busca de Passageiros

Esta mecânica está associada diretamente ao retorno financeiro do jogador. Cabe ao usuário procurar e identificar os passageiros dentro de Mob Town.

3.3.3 Avaliação da Viagem

Após a entrega de um passageiro ao seu destino, o dinheiro que o jogador ganha varia de acordo com a satisfação do mesmo. Esta satisfação é definida pela qualidade e estado do carro durante o transporte.

3.3.4 Condição do Veículo

Ao pilotar o veículo, o jogador pode se envolver em acidentes, seja com outros automóveis, ou em locais estáticos da cidade. Quando as colisões ocorrem, a 'vida' do automóvel diminui, fazendo com que a condição do mesmo fique mais precária. Isso dificulta o controle do veículo, uma vez que a velocidade e a mobilidade variam de acordo com o estado do carro, fazendo com que ele puxe para um lado e fique mais devagar.

No momento em que a vida do carro é zerada, ou seja, quando chega ao 'limite' mínimo, o jogador é impossibilitado de movimentar o táxi e é redirecionado ao mecânico para consertar o seu veículo. O valor do reparo varia de acordo com o estado que o mesmo se encontra, ou seja, quanto mais danificado, mais caro para o jogador.

3.3.5 Upgrades

O fator responsável por deixar a jogabilidade mais fácil para o jogador são os *upgrades*. Em Mob Town existem dois tipos de melhorias: as de *software* e as de *hardware*.

As primeiras são aquelas disponíveis para o celular do jogador e estas facilitam as informações passadas do jogo ao usuário. Já as de *hardware* possuem a função de melhorar seu despenho fisicamente, isto é, deixar mais fácil e rentável a *gameplay*.

3.3.6 Clima Vigente

Algumas mecânicas reagem com o clima vigente da localização real do jogador. A temperatura, por exemplo, pode alterar a maneira com que a IA avalia o seu transporte, podendo variar de forma positiva ou não.

Além disso, as condições meteorológicas afetam o ambiente da *gameplay*, uma vez que a simulação conta com sistemas para alterar o clima dentro do jogo.

4 Desenvolvimento

Para o desenvolvimento de Mob Town foi escolhido o motor[2] de jogos Unity[25], na versão 2020.2.6f1. Optou-se por esta *engine*, pois a mesma possui uma ampla quantidade de usuários e, portanto, possui diversos tutoriais, fóruns para conversas, além de uma documentação bem estruturada. Outro fator decisivo foi que o motor escolhido funciona com a linguagem de programação C#, já dominada pelo autor.

O processo de desenvolvimento de Mob Town contou com a teoria de metodologias ágeis[8]. Uma das etapas deste princípio é a criação de sub entregas, ou seja, dividir o desenvolvimento em partições, onde cada segmento é feito de forma separada para mais tarde realizar a unificação, gerando o produto final. Neste princípio, o 'cliente' é um participante ativo do projeto, onde a cada ciclo da criação ele ajuda na avaliação. Esta dinâmica facilita a remoção, a adição e o reparo de elementos dentro do projeto. Além disso, esse tipo de metodologia ajuda no tempo empregado em cada atividade, pois, se uma ideia falha, neste modelo ela pode ser facilmente removida.

No caso de Mob Town, os clientes eram testadores que, durante o desenvolvimento das etapas, sugeriram melhorias. Corroborando com esta metodologia ágil, o autor optou por utilizar o modelo organizacional de *Scrum*[13], pois o mesmo define prazos para a conclusão de cada etapa, as chamadas *sprints*[3].

A primeira etapa do desenvolvimento foi uma pesquisa procurando os melhores *assets* a serem utilizados no projeto. Buscou-se por modelos relacionados à temática de urbanismo para tornar possível a realização do ambiente da cidade. Foram testadas algumas opções gratuitas, entretanto, estas foram descartadas pois não satisfaziam o critério estipulado, que era possuir uma boa identidade visual[20] e fácil entendimento. Por este motivo, foram selecionados dois *assets* pagos disponíveis na Unity Asset Store[26]. O primeiro, Simple Town - Cartoon Assets[23] e o segundo, Cartoon City Pack - Traffic System[21]. Ambos custaram aproximadamente 30 reais, totalizando em R$60 o valor investido no jogo.

Com o estilo de arte definido, foi montada uma cena de testes, integrando o conteúdo disponibilizado pelas duas aquisições. Nesta cena foi analisada a montagem do cenário com o sistema de tráfico disponibilizado por um dos *assets*. As etapas seguintes foram de desenvolvimento de mecânicas e de sistemas. Estas serão desmembradas durante a seção atual.

4.1 Arte/Gráficos

O gráfico de Mob Town é em 3D, no estilo *lowpoly*[4]. Este tipo de arte foi escolhido pois o autor possui uma maior familiaridade, sendo, então, mais fácil de desenvolver modelos e realizar alterações em *assets* adquiridos. Além disso, o outro motivo da escolha foi o desempenho computacional, uma vez que, segundo o Game Development Stack[19], quanto maior o número de polígonos, maior é o custo de processamento. Essa relação pode

[2]um motor de jogos é um programa que contém diversas bibliotecas e funcionalidades, a fim de facilitar o trabalho do desenvolvedor.

[3]uma *sprint* é um conjunto de tarefas que devem ser executadas em um determinado tempo

[4]Lowpoly é um tipo de modelagem que tem por objetivo diminuir a quantidade de polígonos de um modelo

interferir negativamente na qualidade da execução do jogo, e, por consequência, limitar o uso do mesmo para jogadores que possuem um *hardware* mais potente, o que poderia diminuir o público do jogo.

Como comentado anteriormente nesta seção, foram adquiridos dois *assets*, entretanto, viu-se necessário realizar a modelagem de alguns itens, uma vez que o jogo possui mecânicas muito específicas. Para tal tarefa foi utilizado o *software* de modelagem 3D Autodesk Maya[4], versão 2019. A decisão da escolha por esta aplicação se deve ao fato de que o autor do projeto Mob Town possui familiaridade com a ferramenta, pois já havia desenvolvido *assets* artísticos para outros projetos. Os modelos desenvolvidos foram, em sua maior parte, acessórios para a simulação, tais como: placas de trânsito, postes de luz e diferentes prédios.

Para construir os modelos das placas de trânsito, utilizou-se como referência o padrão visto em algumas cidades reais, ilustrado na figura 5. Além de contribuírem para a simulação, as placas também colaboram com a navegação do jogador pela cidade, uma vez que os nomes das ruas são visíveis durante o jogo.

Figura 5: Placas Trânsito - Fontes: Prefeitura de Itupi & Autor.

Também mostrou-se necessário o desenvolvimento de modelos para alguns elementos da *gameplay*, como o mecânico e passageiros. No primeiro, foi ajustado o modelo de loja de carros de um dos *assets*, pois o mesmo era completamente fechado, sem interior, logo incapacitando o jogador de entrar na loja de reparos. Pensando em corrigir este problema, o modelo em questão sofreu uma adaptação, onde algumas faces foram retiradas, outras criadas, a fim de realizar o interior projetado, como é possível ver na Figura 6.

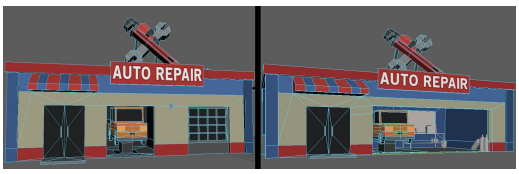

Figura 6: Loja Mecânico antes & depois - Fonte: Autor.

Nos passageiros, mostrou-se necessário após *feedback* dos testadores, a colocação de algo que chamasse a atenção do jogador para a localização de um passageiro. Por este motivo, foi desenvolvido uma estrela que gira em cima do objeto de IA (inteligência artificial), que serve para sinalizar o jogador que o NPC (*non-player-character*) está aguardando seu transporte. Este modelo possui material emissivo, ou seja, seu material irradia cor, desta forma, em ambientes escuros, como o período noturno do jogo, a estrela continua bem visível, cumprindo seu papel em todos momentos da *gameplay*.

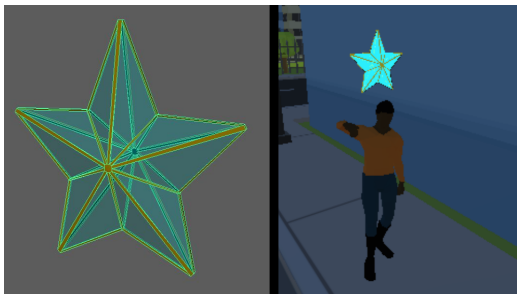

Figura 7: Modelo Estrela - Fonte: Autor.

4.2 Interface Gráfica

A interface gráfica *in-game* de Mob Town procura ser o mais limpa possível, pois além de contribuir para a simulação, também ajuda o jogador a focar nos acontecimentos dentro do jogo. Com este objetivo em mente, foi escolhido colocar as informações em um telefone no lado direito da tela.

Figura 8: Tela de Mob Town com telefone - Fonte: Autor.

Dois motivos principais ajudaram na opção do aparelho digital como objeto de interface. O primeiro é que combina com a simulação, pois motoristas de transporte, hoje em dia, realizam suas operações por meio de celulares, através de aplicações como 99 Táxi, desenvolvido pela empresa 99 Taxi Softwares[1], e Uber, produzido pela Uber Technologies[24]. Outro fator decisivo foi o uso deste objeto no GTA, franquia de jogos explicitada nas inspirações, sendo assim, o método utilizado já passou pelo mercado, dando embasamento prático para o jogo.

A interface do celular é divida entre a tela principal, que contém as informações de temperatura, horário e os ícones dos aplicativos e as telas de cada aplicação, como em um aparelho real. Na Figura11 é apresentado o diagrama de telas do telefone. A arte 2D de cada um dos aplicativos, bem como o *wallpaper*, foram desenvolvidos pelo autor utilizando o software de edição Ilustrator, produzido pela empresa Adobe[2].

Como o jogador possui um sistema de 'vida', foi necessário informar ao mesmo a sua quantidade, utilizando a UI (*User Interface*). A fim de conseguir este objetivo, mas também continuar com o princípio de deixar a tela limpa, decidiu-se que a barra de saúde deveria ficar no centro da tela, em cima do carro, levando em consideração que o foco do jogador, na maior parte da *gameplay*, está no centro do monitor. Entretanto, desta forma, a interface poderia atrapalhar a visão frontal do jogador, implicando em uma possível frustração, então, para evitar isso, a barra de vida aparece na tela somente quando ela precisa ser mostrada, ou seja, quando o jogador de fato perde vida.

Para interface gráfica *out-game*, pensou-se em duas possibilidades. A primeira consistia em utilizar o próprio telefone, colocando as alternativas de menu dentro do aparelho. Já a segunda correspondia a um menu tradicional, ou seja, utilizava a tela cheia como

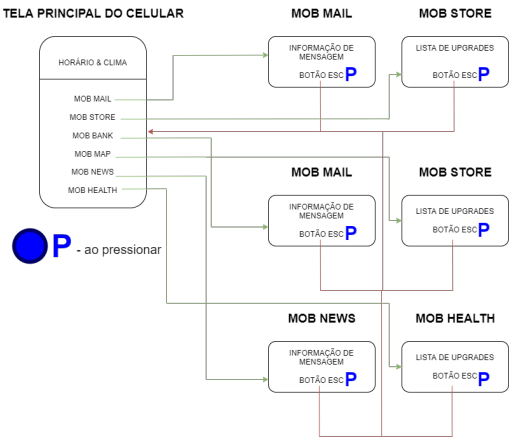

Figura 9: Diagrama de Telas Telefone - Fonte: Autor.

alvo. A decisão entre essas opções se deu analisando o argumento de que se o jogador usa o telefone para simulação, não haveria sentido colocar o menu lá, porque, possivelmente, haveria uma quebra de imersão. Sendo assim, o autor escolheu o menu tradicional como interface *out-game*.

4.3 *Level Design*

A criação do mapa foi feita de forma manual, pois a ideia era se basear em uma cidade real, contando com a disposição de ruas da cidade autônoma de Buenos Aires, e, para isso, foram necessários detalhamentos que são mais fáceis de realizar de forma manual. O autor optou por esta cidade como base porque as ruas se encaixam de forma simétrica, sendo assim, mais fácil de compreender o seu funcionamento, ajudando no aprendizado do jogo.

Além disso, os nomes das ruas possuem relação com o que as mesmas representam, por exemplo: Park Street - esta possui o único parque da cidade. Desta forma, o mapa de jogo tende a colaborar com a curva de aprendizado do jogo.

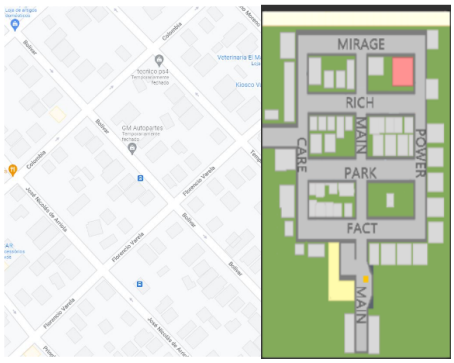

Figura 10: Mapas - Cidade autônoma & Mob Town - Fonte: Google Maps e Autor.

4.4 Trilha e Efeitos Sonoros

Para a etapa de trilha sonora, foi realizada uma pesquisa, a fim de procurar músicas gratuitas, com licença e que combinassem com o jogo. Alguns sites conhecidos foram buscados, como Sound Bible, Game Dev Market, Free Sound, Ben Sound. No entanto, os áudios utilizados no projeto foram encontrados no Youtube Studio[29]. A ferramenta possui uma grande variedade de efeitos e trilhas de forma gratuita, além de não possuir problemas com direitos autorais[11].

O jogo conta com uma trilha que roda em *loop* durante a *gameplay*, além de efeitos sonoros que tornam a experiência do usuário mais imersiva. Barulhos de chuva, de trovão, de buzina, de motor, de pessoas falando na rua, de cachorros latindo, além de sons para cada ação do celular (mover, notificações e etc) aperfeiçoam a interação do jogador.

Os efeitos sonoros baixados foram modificados utilizando a aplicação de edição de vídeos e sons Da Vinci Resolve, versão 2017[7]. Nela foi feita a unificação de efeitos e alteração de parâmetros de cada áudio para elaborar os sons.

4.5 Estética

A estética de um jogo digital é um quesito muito importante, pois ela aumenta a sensação de imersão, segundo o artigo de Eduardo Müller[17]. A ´beleza´ estética faz com que o usuário sinta de forma mais realista as emoções que o jogo propõe, gerando prazer. Isso tudo é possível graças à compatibilidade com o tema proposto, uma consistência de todos elementos presentes, entre eles áudio, arte, mecânicas, ambientação, etc.

Os aspectos artísticos de Mob Town, anteriormente descritos nesta seção, refletem o conceito de estética abordado acima, uma vez que todos os elementos do jogo combinam entre si. Por possuir gráficos não realistas, a interface também não é realista, da mesma forma que os efeitos sonoros e etc. Além disso, é possível ver o conceito de estética no logo do projeto. Como a temática proposta por Mob Town é mobilidade urbana, optou-se em fazer uma analogia ao tema no logotipo do projeto, onde é possível ver elementos urbanos enfeitando e substituindo letras.

4.6 Tecnologias Utilizadas

Nesta subseção será abordado o desenvolvimento de alguns sistemas, mecânicas e integrações.

4.6.1 Sistema Dia/Noite e Horário

A fim de fazer o ciclo entre dia e noite, foi necessário a criação de dois objetos emissores de luz direcional[5], cada um contendo suas informações específicas, como, por exemplo, intensidade e cor. Desta forma foi possível começar a simulação do sol e da lua respectivamente.

Para dar vida ao 'sol' e a 'lua' foi criado um *script* que possui a tarefa de rotacionar os objetos de acordo com o tempo e modificar seus atributos. No código foram criadas três variáveis responsáveis pelo controle do tempo no jogo. A primeira, *dayLenght*, é responsável pelo tamanho total do dia em segundos. A segunda, *time*, atua como controladora do horário atual em tempo de execução. Já a terceira, *timeRate*, é o valor a ser acrescentado ao horário a cada *frame* do jogo. Vale dizer que este valor varia de acordo com o extensão do dia, que, em Mob Town, dura 1.200 segundos (20 minutos). Sendo assim, a cada quadro, o jogo altera seu tempo de acordo com a duração total escolhida.

As fontes de luz rotacionam com o uso da variável *time* e de forma inversamente proporcional, ou seja, enquanto uma está no topo do mundo, a outra está em baixo. Com isso é obtido a simulação do efeito de rotação da terra, porém, neste caso, com a 'terra' no centro do sistema solar.

Após esse processo, é feito o uso das chamadas *Animation Curves*. Essa é uma ferramenta disponibilizada pelo motor do jogo, que permite a criação de curvas de animação e armazenamento de pontos chave, que podem ser alterados ao longo do tempo[27].

A intensidade da luz de cada um dos objetos emissores é feita utilizando essas curvas de animação. No caso, o tempo levado para

[5]uma luz direcional aponta somente para uma direção e todos objetos na cena são iluminados por esta fonte.

percorrer a curva é igual ao período do jogo, ou seja, a intensidade da luz varia de acordo com os valores da curva, que, por sua vez, mudam de acordo com a variável *time*. O mesmo princípio é aplicado para a mudança de cor das luzes, entretanto, ao invés de curvas de animação, é utilizado um gradiente, que também opera seus valores com base no tempo.

No momento em que a intensidade do sol fica quase zerada, o objeto solar é desativado, o estado do sistema passa a ser noturno[6] e a lua é ativada. Esta opera da mesma maneira, mas quando sua intensidade é zerada, o sol é ativado e o estado do sistema passa a ser diurno.

Para obter um melhor resultado de iluminação, os atributos da *pipeline* gráfica, intensidade do ambiente e reflexão são alterados, utilizando o mesmo sistema de curvas previamente descrito. A fim de passar o tempo normalizado para o relógio real, é feito um cálculo que modifica seu valor para horas e minutos. Esta operação é divida em três partes.

A primeira consiste em passar o valor normalizado para hora, multiplicando este número por 24, pois um dia contém 24 horas. A segunda é passar a hora para minutos, ou seja, multiplicar o valor resultante da primeira operação por 60, uma vez que em uma hora existem 60 minutos. A última etapa é uma divisão entre o valor total em minutos por 60, onde o resto/sobra da operação é o valor em minutos do jogo.

Depois, esse valor de tempo é passado para HUD, para, então, ser mostrado ao jogador.No momento em que o tempo chega ao seu valor máximo, o mesmo é resetado para zero, reiniciando, assim, o ciclo.

4.6.2 Luzes da Cidade

Para colaborar com a simulação desejada, foi criado um sistema cuja responsabilidade é ligar as luzes da cidade no momento em que estiver de noite. Para obter esse resultado, foi criado um vetor contendo todos os objetos de luz da cena (postes e faróis dos automóveis).

O código monitora o estado do dia (descrito no sistema anteriormente explicado). Quando o mesmo é alterado para noite é operado um *loop*, que possui a função de iterar para cada posição no vetor de fontes de luz, ativando o objeto em questão. O mesmo vale para apagar as luzes, porém de forma inversa, ou seja, o *loop* desativa os objetos no momento em que o estado do dia for diurno.

4.6.3 Sistema Clima - OpenWheatherMap

A OpenWheatherAPI[15] é uma plataforma cuja funcionalidade é prover dados climáticos para os aplicativos que a utilizam. Sua principais concorrentes são ClimaCell API e Wheatherbit, segundo o blog Last Call[14]. O blog mencionado avaliou cerca de 22 API´s e nele consta, na primeira posição, a plataforma utilizada neste projeto. Por ser gratuita até 1 milhão de chamadas é uma boa opção para fins de projeto final. Suas funcionalidades variam de acordo com o valor de pagamento, sendo que na forma gratuita temos o histórico climático, clima atual, poluição do ar e alertas climáticos suprindo todas as necessidades do projeto.

Para utilizar o clima local do usuário é preciso saber sua localização, para então passar esse dado para API. A API utilizada funciona de duas maneiras, a primeira com o nome da cidade desejada e, a segunda com as coordenadas geográficas de latitude e longitude. No Mob Town, foi utilizada a primeira opção, pois para obter a localização do usuário, em coordenadas geográficas na plataforma de computador, é necessário acesso ao seu endereço de IP(Internet Protocol)[7] e isso, em alguns casos, não é visto como uma boa prática, uma vez que pode acarretar em uma vulnerabilidade (no caso deste projeto, seria necessária a participação de um serviço terceirizado, que pode não ser 100% seguro).

Sendo assim, foi realizado um menu de *dropdown* com algumas cidades pré-definidas, cabendo ao jogador escolher sua cidade no início da *gameplay*. Com a localização definida, quando o jogo começa é feito um *Web Request*, isto é, um chamado na internet pedindo as informações da API. Para fazer este chamado é necessário gerar uma URL, que deve conter os padrões da API como parte de seu corpo, além de conter também a informação do nome da cidade obtido anteriormente.

O retorno desta operação é um arquivo JSON, que é lido, interpretado e possui seus dados armazenados em um objeto dinâmico, que possui a função de informar para os outros *scripts* a situação meteorológica do jogador. Após a primeira interação com a API, um contador de 2 minutos opera e, quando chega ao fim, é refeito o chamado, para, então, atualizar o clima caso haja alguma alteração.

4.6.4 Chuva e Nuvens

A chuva e as nuvens no jogo são elementos visuais criados para deixar mais imersivas a mecânica do clima e a própria cidade. Para criar as nuvens, foi feito uso da ferramenta de criação de *shaders*, Shader Graph[], disponibilizada pelo motor do jogo.

Dentro da ferramenta, utilizou-se o nodo *Simple Noise*, um efeito de ruído onde são criadas distorções aleatórias. Esse efeito é animado com uma velocidade, dando a ideia de movimento das nuvens. Além disso, para controlar a densidade do material, foi criado um nó de poder(*Power Node*). A superfície do *shader* é configurada para possuir transparência, logo, sua posição *alpha* recebe o *output* do nó de poder, fazendo com que partes mais escuras da textura não apareçam.

Para simular a formação e distorção das nuvens, foi criado um outro nodo de ruído, este possuindo valores de velocidade da animação diferentes do anterior. Com isso, ambos os ruídos são multiplicados antes de passarem para o *Power Node*, criando, assim, uma outra camada de aleatoriedade para o *shader*, evitando o reconhecimento do padrão das nuvens.

Esse *shader* foi criado e colocado em um plano curvo para impedir que, nas bordas do mundo, não houvessem nuvens, uma vez que se fosse um plano reto, não haveria como fazer o horizonte possuir nuvens. Para deixar as nuvens ainda mais 'fofas', as áreas de ruído modificam os vértices do modelo, fazendo com que as nuvens tomem uma forma volumétrica.

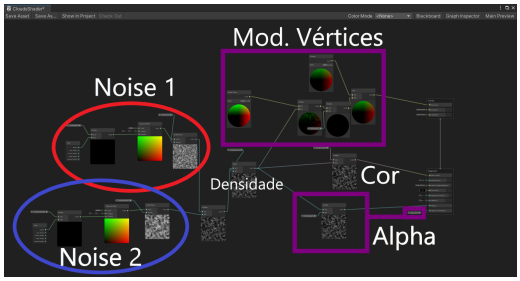

Figura 11: Shader Nuvens - Shader Grpah - Fonte: Autor.

Para a simulação da chuva foi criado um sistema de partículas em forma de caixa que segue o jogador. Esse sistema emite partículas esticadas e finas, em direção ao solo. Por possuírem esta forma, em grande quantidade acabam sendo similar as chuvas reais. Para deixar mais singelo, mudou-se a transparência e a cor, uma vez que a chuva não é opaca.

Esse sistema conta com a detecção de colisão para cada partícula, pois, para possuir maior veracidade, viu-se necessário a adição de respingos e marcação d´água nos pontos que as gotas caem. Para

[6] o código conta com uma variável responsável por mudar o estado do dia/noite

[7] o endereço IP identifica a localização de uma rede na internet

obter esse resultado, foi feito o uso de *subemitters*, sistemas de partículas filhos que são chamados quando requisitados pelo pai. Esse chamado depende de alguma condição, que, no caso, é a colisão das partículas do pai. Logo, quando esta subpartícula é chamada, ela libera três ou mais gotículas de água aleatórias, que possuem uma vida curta mas cumprem o seu propósito.

Da mesma forma operam os marcadores de água, porém, ao invés de sumirem imediatamente, eles deixam de aparecer diminuindo seu valor de transparência com o tempo, para só então serem deletados. Para fazer a sincronia deste sistema visual com o clima real do jogador, criou-se um *script* que possui a tarefa de verificar o conteúdo da variável que armazena a condição climática do jogador. Se a variável estiver indicando chuva/tempestade, o código altera os valores de densidade e cor do*shader* para algo mais chuvoso, ou seja, cinzento e denso, ativando também o sistema de partículas anteriormente mencionado. Caso contrário, os valores mudam para sua configuração inicial, que é definida na criação do *shader*.

É importante mencionar que a alteração visual ocorre de forma gradual, porém, rápida, utilizando a função *Lerp()*, disponibilizada pelo motor. A operação mencionada interpola linearmente dois valores durante um determinado período de tempo. Desta forma, o valor de densidade é modificado gradativamente.

4.6.5 Pedestres e Passageiros

Os pedestres e os passageiros da *gameplay* foram feitos utilizando princípios de inteligência artificial. A principal base para esta operação é o conceito de Máquina de Estados. Essa ideia consiste em dividir o objeto em estados lógicos, fazendo com que a cada situação, um novo tipo de comportamento seja exibido.

Em Mob Town, os estados são utilizados para modificar a atitude dos pedestres. Cada civil possui a habilidade de caminhar até seu *waypoint* e de estar na sua última volta pela cidade. Somente uma pessoa por vez pode se tornar um passageiro e apenas essa pessoa escolhida aleatoriamente possui a habilidade de ir até o táxi, evitando assim, que o jogador pegue múltiplos passageiros ou se confunda com seu objetivo. Na Figura 12, é apresentado o diagrama da *State Machine*.

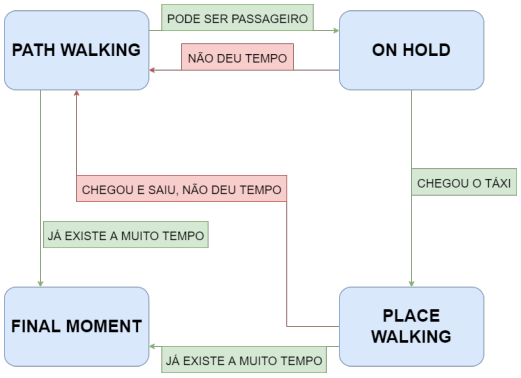

Figura 12: Diagrama da Máquina de Estados de Mob Town - Fonte: Autor.

O estado 'caminhando', em código, é chamado de *PathWalking* e, sua lógica consiste em encontrar um caminho até seu ponto de destino. Quando o agente chega ao local, um novo ponto meta é sorteado e o ciclo se repete.

Para fazer o *Pathfinding*, isto é, a escolha do caminho, foi feito uso da ferramenta disponibiliza pelo motor do jogo, *NavMesh*. Esse mecanismo permite a criação de uma malha de navegação gerada pelo motor, com as definições criadas pelo desenvolvedor do jogo. No projeto, a malha foi gerada nas calçadas, fazendo com que os agentes andassem somente onde é permitido. Para atravessar a rua, criou-se os chamados *NavMesh Links*. A função desta *feature* é lincar os *navmeshs*, fazendo com que, no momento em que um agente chegar a um link, ele vá para a localização que o mesmo aponta.

O civil selecionado recebe a liberação para passar do estado 'caminhando' para o estado de 'aguardo'. A chance de ocorrer esta mudança de lógica é de 60%, caso ela não aconteça, um contador, cujo valor de inicialização é randômico, opera e, quando sua quantia for zerada, após alguns instantes, o sorteio é refeito com a mesma quantidade de chance. Caso o NPC entre no estado *OnHold*, o objeto da estrela e o colisor responsável por detectar a presença do *player* são ativados. Além disso, um outro contador randômico opera e enquanto seu valor for maior do que zero, a IA continua no mesmo estado aguardando pelo jogador.Entretanto, no momento em que o valor é zerado, o estado muda novamente para *PathWalking*. Desta maneira, o ciclo é repetido até o jogador pegar e entregar o passageiro.

Quando o táxi se aproxima do NPC, que está no aguardo, o estado da IA muda para *PlaceWalking*, fazendo com que o objeto inteligente vá em direção ao veículo e entre nele, liberando, então, a informação de destino escolhido. Para escolher um destino, a máquina sorteia uma quantia cujo valor máximo é o tamanho do vetor de destinos. Este *array* possui, em sua composição, todos pontos que um passageiro pode querer ir. Com a decisão do local alvo, a IA retira as informações de local e nome do destino, e as transfere para o HUD, que, por sua vez, informa o jogador.

No momento em que o passageiro é entregue no seu destino, seu estado volta a ser de caminhar e sua capacidade de virar passageiro é removida. Na sequência, um novo civil com a habilidade de ser transportado é criado no mundo. Depois, o jogador recebe seu pagamento e vai em direção ao outro passageiro.

4.6.6 Jogador - Táxi

A simulação física da movimentação do veículo controlado pelo jogador utiliza o conceito criado pela *engine* de *WheelCollider*. Para controlar essa ferramenta, foi criado um sistema que armazena o *input* do jogador em variáveis.

A movimentação vertical do táxi se faz pela multiplicação da variável de *input* vertical[8], com um valor de velocidade predefinido pelo autor. O resultado dessa operação é passado ao torque das duas rodas da frente do automóvel. Já para o movimento horizontal, é passado ao valor de dobragem das rodas frontais, o valor de entrada horizontal[9], multiplicado com um ângulo máximo de dobra das rodas.

Para a frenagem do veículo é verificado se o usuário está de fato pressionando a tecla de freio. Se esta condição for verdadeira, aplica-se um valor de frenagem ao torque de freio das quatro rodas.

O automóvel possui um sistema de vida que é solicitado toda vez que ocorre alguma colisão. O valor de dano obtido pelo táxi varia dependendo da velocidade do mesmo no momento do incidente. O cálculo de dano possui o valor da velocidade atual em km/h dividido por dois, ou seja, o valor de dano equivale a metade da velocidade atual, desta forma, variando de acordo com a agilidade do carro.

5 TESTES

O processo de teste de Mob Town começou no dia 22 de outubro de 2021. O teste desta versão, chamada de versão 0.7, terminou no dia 28 de novembro do mesmo ano, totalizando os 38 dias desta fase do projeto. Dentro desse período, a página do jogo, no site Itch.io[12], obteve um total de 65 acessos, com 20 *downloads*, conforme ilustra a Figura 13

A grande maioria dos testes foram realizados de forma *online*/remota, através de um formulário disponibilizado na página

[8]este *input* atribui valores para as teclas W e S, desta forma é possível saber se o usuário quer ir para frente ou para trás, respectivamente

[9]este *input* atribui valores para as teclas A e D, desta forma é possível saber se o usuário quer ir para direita ou para esquerda

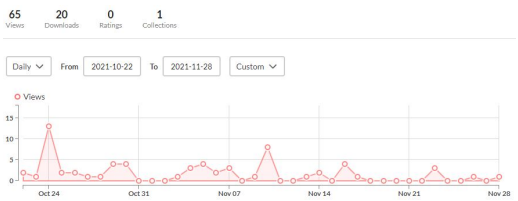

Figura 13: Acessos na página de Mob Town na plataforma itch.io - Fonte: Autor.

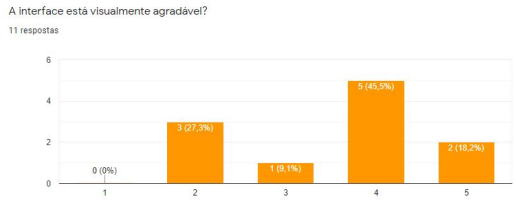

Figura 15: Resumo Respostas Interface Gráfica - Fonte: Autor.

do jogo. Entretanto, em alguns casos, foram realizadas entrevistas apreciativas utilizando a plataforma Discord[9], buscando, por meio do diálogo com o jogador, formas de solucionar brechas e falhas na *gameplay*.

Em sua maioria, os *testers* eram do sexo masculino e possuíam entre 20 e 30 anos. Os jogadores que preencheram o formulário não foram obrigados a se identificar para não prejudicar a veracidade dos testes. Foram feitas 20 perguntas objetivas e dissertativas, divididas em jogabilidade e arte/estética. Foram obtidas 11 respostas pelo formulário e o restante dos *testers* fizeram entrevista com o autor.

6 Análise de Resultados

Como comentado anteriormente, um total de 11 jogadores responderam o formulário disponibilizado na página do projeto. Quando questionados sobre a compreensão dos objetivos de Mob Town, 73% disseram que não é difícil entender as metas do jogo.

Houve uma divergência de opiniões grande quanto ao controle do táxi, como ilustra a Figura 14. Alguns usuários disseram preferir o freio do veículo na tecla S, junto com a ré. Outros comentaram e afirmaram que o *SHIFT* era a melhor opção. E, ainda, alguns preferiram utilizar a barra de espaço para função.

Figura 14: Pegunta sobre comando da tecla de freio - Fonte: Autor.

Por esse motivo, surgiu a ideia de possuir uma tela, onde o usuário possa definir qual entrada ele quer para o freio, uma vez que essa mostrou ser uma definição pessoal.

Tanto as entrevistas apreciativas, quanto os formulários, mostraram que existe uma lacuna durante a entrega de um passageiro. Essa lacuna é referente a informação do destino do passageiro. Muitos testadores apontaram que durante a entrega, esqueceram aonde deveriam levar o indivíduo, e, desta forma, entraram em uma espécie de confusão e inquietação.

Todos os jogadores disseram compreender a função de cada um dos aplicativos do telefone. Além disso, 72,8% dos usuários declararam que a interface do jogo estava boa ou ótima, de acordo com a Figura 15.

Ainda segundo o formulário, 81% dos testadores asseguraram que os elementos do jogo representam uma cidade de fato. Esse dado corrobora com a reação vista nas entrevistas apreciativas, uma vez que a maioria dos entrevistados elogiaram o *level design* e a arte do jogo.

7 Considerações Finais

De forma geral, conclui-se que Mob Town atinge seu objetivo de simular um ambiente urbano e que os elementos trazidos de jogos referência foram importantes para a *gameplay*, pois incrementaram a diversão do jogo. Do mesmo jeito, o uso do clima do jogador como entrada mostrou ser um diferencial, pois os jogos do gênero apresentados não utilizam esse tipo de *feature*. Além disso, a mecânica colaborou com a simulação e incrementou a dinâmica e a imersão do jogo.

A colocação do jogador de forma ativa e prática no desenvolvimento da economia de sua empresa corroborou com o objetivo do gênero *tycoon*. As pesquisas feitas durante o desenvolvimento foram de suma importância no resultado final, pois trouxeram embasamento prático e teórico para o projeto, ajudando o autor na tomada de decisões ao decorrer da criação do jogo.

7.1 Trabalhos Futuros

O projeto Mob Town mostrou ter melhorias a serem feitas, que variam desde a adição de mecânicas até a evolução da ideia para algo novo no jogo. Novos *upgrades*, tanto de *software* quanto de *hardware* são melhorias a serem acrescentadas. Uma maior interação do jogador com elementos da cidade também é algo interessante de se aperfeiçoar. A expansão do cenário, em termos de tamanho, poderá ser adicionada nas próximas atualizações do projeto.

O jogo pode possuir, também, uma segunda camada de *gameplay*. Esta seria como um *tycoon* convencional, ou seja, o jogador passaria de motorista, para somente CEO e, o estilo de jogabilidade seria outro, bem como as aquisições a serem feitas e os desafios, sempre levando em conta o progresso feito com a jogabilidade anterior. Além dos quesitos práticos, o jogo ainda precisa ser publicado e monetizado, contendo as características comentadas anteriormente neste artigo vistas na Seção 2.3.

Agradecimentos

Agradeço a todos os meus amigos, professores e familiares que ajudaram o desenvolvimento deste projeto. Em especial, gostaria de a agradecer minha irmã, Juliana, que em diversos momentos colaborou com o desenvolvimento deste artigo e que me apoiou emocionalmente durante o processo de criação do jogo. Obrigado a todos aqueles que testaram o jogo e retornaram *feedbacks*, seja pelo formulário ou em contato direto.

Reconheço, em especial, o esforço dos meus amigos: Gustavo, Rodrigo C., Pedro, Marco, Gabriel, André, Rodrigo W. e Matheus, que testaram o jogo durante todo o processo de desenvolvimento.

Agradeço a minha mãe, Fernanda, e a minha avó, Leide, que me deram confiança emocional durante a criação do projeto. Obrigado meu avô, Roberto, que durante toda formação acadêmica me apoiou e celebrou minhas conquistas. Este projeto possui grande influência sua!

Referências

[1] 99Taxi. Site. `https://99app.com/`. Acesso em: 26 nov. 2021.

[2] Adobe. Site. https://www.adobe.com/creativecloud/photography/discover/chromatic-aberration.html. Acesso em: 23 nov. 2021.

[3] AppStore. Site. https://apps.apple.com/br/app/roblox/id431946152. Acesso em: 28 nov. 2021.

[4] Autodesk. Site. https://www.autodesk.com.br/products/maya/overview?panel=buy&AID=12904993&PID=8299320&SID=jkp_EAIaIQobChMI1Jv-jr229AIVkRh9Ch0JigtuEAAYASAAEgIi1vD_BwE&cjevent=4960fbd34ed811ec801f165d0a82b82d&affname=8299320_12904993. Acesso em: 26 nov. 2021.

[5] L. Chieppe. Monetização para jogos. https://edisciplinas.usp.br/pluginfile.php/5751053/mod_resource/content/1/Aula\%2010\%20-\%20Monetizacao.pdf. Acesso em: 27 nov. 2021.

[6] R. Corporation. Site. https://www.roblox.com/. Acesso em: 22 nov. 2021.

[7] DaVinciResolve. Site. https://www.blackmagicdesign.com/products/davinciresolve/?gclid=EAIaIQobChMIqoi5sfO79AIVMQV9Ch29wQeHEAAYASAAEgJMF_D_BwE. Acesso em: 28 nov. 2021.

[8] F. N. de Oliveira. Scrum como metodologia Ágil na produção de jogos digitais. https://www.fabricadejogos.net/posts/scrum-como-metodologia-agil-na-producao-de-jogos-digitais/. Acesso em: 22 nov. 2021.

[9] Discord. Site. https://discord.com/. Acesso em: 28 nov. 2021.

[10] R. Games. Site. https://www.rockstargames.com/br/. Acesso em: 22 nov. 2021.

[11] Google. Site. https://support.google.com/youtube/answer/3376882?hl=pt-BR. Acesso em: 28 nov. 2021.

[12] Itch.io. Site. https://itch.io/. Acesso em: 28 nov. 2021.

[13] C. Keith. *Agile Game Develpment With Scrum*. Pearson Education, first edition, 2010.

[14] LastCall. Site. https://rapidapi.com/blog/access-global-weather-data-with-these-weather-apis/. Acesso em: 27 nov. 2021.

[15] O. W. Map. Site. https://openweathermap.org/. Acesso em: 22 nov. 2021.

[16] Morningstar. Site. https://www.morningstar.com/. Acesso em: 19 nov. 2021.

[17] E. F. Müller. Os conceitos estÉtico-visuais dos jogos digitais. *PUCRS*, 1(1):29–134, July 2011.

[18] A. L. Pereira. Indústria de games movimentou mais de us$ 120 bilhões em 2019. https://www.tecmundo.com.br/cultura-geek/148956-industria-games■movimentou-us-120-bilhoes-2019.htm/. Acesso em: 20 out. 2020.

[19] Philip. Game dev site. https://gamedev.stackexchange.com/questions/82538/why-are-huge-polygon-amounts-bad. Acesso em: 23 nov. 2021.

[20] PontoDesign. Site. https://www.pontodesign.com.br/entenda-como-criar-uma-identidade-visual-adequada-para-sua-marca/#:~:text=\%C3\%89\%20o\%20conjunto\%20de\%20todos,gerado\%20a\%20partir\%20desses\%20elementos. Acesso em: 19 nov. 2021.

[21] M. Sarishvili. Asset. https://assetstore.unity.com/packages/3d/environments/urban/cartoon-city-pack-simple-traffic-system-18876. Acesso em: 26 nov. 2021.

[22] Steam. Site. https://store.steampowered.com/app/613880/Zoo_Tycoon_Ultimate_Animal_Collection/. Acesso em: 22 nov. 2021.

[23] SyntyStudios. Asset. https://assetstore.unity.com/packages/3d/environments/urban/simple-town-cartoon-assets-43500. Acesso em: 26 nov. 2021.

[24] Uber. Site. https://www.uber.com/br/pt-br/. Acesso em: 26 nov. 2021.

[25] Unity. Site. https://docs.unity3d.com/Manual/index.html. Acesso em: 22 nov. 2021.

[26] UnityAssetStore. Site. https://assetstore.unity.com/?gclid=EAIaIQobChMI6P-3sLm29AIVnB-tBh1trg9lEAAYASAAEgJOT_D_BwE&gclsrc=aw.ds. Acesso em: 26 nov. 2021.

[27] UnityDoc. Site. https://docs.unity3d.com/Manual/animeditor-AnimationCurves.html. Acesso em: 28 nov. 2021.

[28] D. Williams. Structure and competition in the u.s. home video game industry. *IEEE Transactions on Visualization and Computer Graphics*, 4(1):44–54, June 2002.

[29] YoutubeStudio. Site. https://studio.youtube.com/. Acesso em: 28 nov. 2021.