UNIVERSIDADE FEDERAL DO RIO DE JANEIRO
INSTITUTO DE MATEMÁTICA
CURSO DE BACHARELADO EM CIÊNCIA DA COMPUTAÇÃO

ALLAN MONTEIRO DAVID
MIGUEL PÉRES DE CASTRO

UMA ABORDAGEM PARA ANÁLISE DE QUALIDADE DE JOGOS BASEADA EM TELEMETRIA: MODELO E IMPLEMENTAÇÃO NA UNITY

RIO DE JANEIRO
2019

ALLAN MONTEIRO DAVID
MIGUEL PÉRES DE CASTRO

UMA ABORDAGEM PARA ANÁLISE DE QUALIDADE DE JOGOS BASEADA EM TELEMETRIA: MODELO E IMPLEMENTAÇÃO NA UNITY

Trabalho de conclusão de curso de graduação apresentado ao Departamento de Ciência da Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Orientador: Prof. Geraldo Bonorino Xexéo, D.Sc

RIO DE JANEIRO
2019

David, Allan Monteiro

D249a Uma abordagem para análise de qualidade de jogos baseada em telemetria: modelo e implementação na Unity / Allan Monteiro David, Miguel Péres de Castro. – 2019.

55 f.

Orientador: Geraldo Bonorino Xexéo.

Trabalho de Conclusão de Curso (Bacharelado em Ciência da Computação) - Universidade Federal do Rio de Janeiro, Instituto de Matemática, Bacharel em Ciência da Computação, 2019.

1. Jogo. 2. Desenvolvimento. 3. Telemetria. 4. Qualidade. I. Castro, Miguel Péres. II. Xexéo, Geraldo Bonorino (Orient.). III. Universidade Federal do Rio de Janeiro, Instituto de Matemática. IV. Título.

ALLAN MONTEIRO DAVID
MIGUEL PÉRES DE CASTRO

UMA ABORDAGEM PARA ANÁLISE DE QUALIDADE DE JOGOS BASEADA EM TELEMETRIA: MODELO E IMPLEMENTAÇÃO NA UNITY

Trabalho de conclusão de curso de graduação apresentado ao Departamento de Ciência da Computação da Universidade Federal do Rio de Janeiro como parte dos requisitos para obtenção do grau de Bacharel em Ciência da Computação.

Aprovado em ___ de _____ de _____

BANCA EXAMINADORA:

Prof. Geraldo Bonorino Xexéo,
D.Sc. (COPPE/UFRJ)

Prof. Eduardo Freitas Mangeli de Brito,
M.Sc (UFRJ)

Profa. Adriana Santarosa Vivacqua,
D.Sc. (UFRJ)

AGRADECIMENTOS

Allan Monteiro David

Quero agradecer a todas as pessoas especiais que conheci nesse trajeto, aprendi muito com todas elas.

Ao meu amigo Miguel que topou fazer esse trabalho comigo e que me ajudou em vários momentos com uma cerveja.

Aos meus pais, Laura e Alexandre, que sempre me deram suporte e se preocuparam em me fazer seguir adiante.

AGRADECIMENTOS

Miguel Péres de Castro

Agradeço aos meus colegas do curso, por todas as rodas de estudo e discussões que enriqueceram meu aprendizado.

Ao meus amigos, por todas os momentos de lazer proporcionados fora do ambiente acadêmico.

Ao Allan, por ter me acompanhando por todo o trajeto, desde o começo.

E sou especialmente grato à minha família, pelo suporte constante e incentivo para sempre seguir em frente independente das adversidades.

RESUMO

Telemetria (do grego “tele” significa remoto e “metron” medida) refere-se à uma forma de captação de dados que permita uma medição a distância. No desenvolvimento de jogos, seu uso é muito importante para entender o comportamento dos jogadores e na identificação do impacto de uma mudança no jogo (como um novo item ou cenário adicionados). Os dados obtidos ajudam os desenvolvedores a terem insumos para a melhora do “game design”. O objetivo deste trabalho é apresentar uma ferramenta de telemetria *open source* para auxiliar na verificação de qualidade em jogos, através da captação e visualização de métricas relevantes para o jogo em questão.

Palavras-chave: Jogo. Desenvolvimento. Telemetria. Qualidade.

ABSTRACT

Telemetry (from the greek "tele" means remote and "metron" means measure) refers to the process of remote data capture/measurement. In game development its usage is really important to understand player behavior and to identify impact on design changes (like the addition of a new level or item to a game). The data gathered can aid developers to make better choices regarding their game's design. The objetive of this project is to present an open source telemetry tool to capture and visualize metrics, to ensure that the decisions taken during development will make a good impact in the quality of the final product.

Keywords: Game. Development. Telemetry. Quality.

LISTA DE ILUSTRAÇÕES

LISTA DE CÓDIGOS

LISTA DE ABREVIATURAS E SIGLAS

2D	Bidimensional
3D	Tridimensional
AAA	“Triple A”: jogos de alto orçamento e campanha promocional
API	Application Programming Interface
BaaS	Backend-as-a-Service
DAU	Daily Active Users
JSON	JavaScript Object Notation
NPC	Non-Playable Character
OPM	Open Provenance Model
QA	Quality Assurance
REST	Representational State Transfer
URL	Uniform Resource Locator
WWDC	Worldwide Developers Conference

SUMÁRIO

1 INTRODUÇÃO

A indústria dos videogames começou a tomar forma durante os anos 70. Hoje, é uma das maiores no ramo de entretenimento, tendo movimentado mais de 100 bilhões de dólares em 2017. Nessa indústria existem muitas empresas grandes e bem consolidadas, porém é cada vez mais comum a presença de pequenas empresas, chamadas de *"indies"*, às vezes composta por apenas uma pessoa.

1.1 MOTIVAÇÃO

Mesmo com poucos recursos financeiros empresas menores conseguem marcar presença apostando em inovação para chamar a atenção do público e disputar o mercado. Porém, no aspecto de qualidade, diferente das grandes empresas, que conseguem ter equipes dedicadas apenas a testarem os jogos e verba para adquirir ferramentas de análise complexas, as independentes precisam recorrer a outras formas de fazer a verificação de qualidade. Uma dessas maneiras é disponibilizar o jogo, logo numa fase inicial, para um público selecionado, a fim de terem um parecer sobre a qualidade o mais cedo possível, vindo direto de seu público. Esse trabalho pretende ajudar esses desenvolvedores fornecendo para eles uma maneira de verificar a qualidade de seus jogos de uma maneira simples e barata.

1.2 OBJETIVO

Este trabalho tem como objetivo apresentar uma ferramenta de telemetria para auxiliar na verificação de qualidade em jogos. Com ela é esperado que desenvolvedores consigam colher mais dados sobre como os jogadores interagem com seus jogos e com isso gerar insumos para a melhorar a qualidade do seu produto.

1.3 ESTRUTURA DO TEXTO

Este texto teve sua estrutura dividida da seguinte maneira: no capítulo 2 (Referências Teóricas) são definidos conceitos necessários para o entendimento integral do projeto; o capítulo 3 (Tecnologias) descreve tecnicamente como o projeto foi implementado; O capítulo 4 (Proposta de Modelo) detalha o modelo, descrevendo os tipos de evento e suas características e as métricas disponíveis; O capítulo 5 (Manual do Usuário) descreve como a ferramenta deve ser configurada e utilizada, explicando o uso dos diferentes tipos de evento. O capítulo 6 (Exemplos) mostra a ferramenta em ação, sendo aplicada em dois tipos diferentes de jogos; e por fim o capítulo 7 expõe as conclusões do projeto.

2 REFERÊNCIAS TEÓRICAS

2.1 DEFINIÇÕES

Nesta seção, serão apresentadas definições necessárias para o entendimento das características de um jogo que melhor se beneficiaria da ferramenta desenvolvida neste trabalho.

2.1.1 Jogos

No dicionário Michaelis, a primeira definição dada a palavra jogo é:

> Qualquer atividade recreativa que tem por finalidade entreter, divertir ou distrair; brincadeira, entretenimento, folguedo.

Já entre estudiosos e filósofos parece não existir um consenso e existem muitas definições para o termo "jogo" ou "jogar", dentre as quais algumas se destacam, como a dada por Johan Huizinga em "Homo Ludens" (HUIZINGA, 2017):

> [...] uma atividade ou ocupação voluntária, exercida dentro de certos e determinados limites de tempo e de espaço, segundo regras livremente consentidas, mas absolutamente obrigatórias, dotado de um fim em si mesmo, acompanhado de um sentimento de tensão e de alegria e de uma consciência de ser diferente da 'vida cotidiana'

Outra definição bem aceita é a de Katie Salen e Eric Zimmerman no livro "Rules of Play" (SALEN; ZIMMERMAN, 2003) que diz que um jogo é:

> [...] um sistema no qual o jogador engaja em um conflito artificial, definido por regras, com resultados quantificáveis

Aqui usaremos a definição dada por Elliot Avedon e Brian Sutton-Smith (AVEDON; SUTTON-SMITH, 2015):

> Jogos são um exercício de sistemas de controle voluntário, em que há uma competição entre forças, limitadas por regras para produzir um resultado desequilibrado.

Essa definição está alinhada com a do Johan Huizinga ao dizer que a atividade deve ser voluntária. Assim como concorda com a da Katie Salen quando define que deve existir um conflito e resultado no final. Todas as três definições seguem a premissa em comum de que devem existir regras bem definidas.

2.1.2 Jogos Eletrônicos

Nicolas Esposito, no artigo "A Short and Simple Definition of What a Videogame Is" (ESPOSITO, 2005), dá uma definição simples sobre o que é um videogame, nela ele diz:

> Um video game é um jogo no qual jogamos por meio de um aparato audiovisual e o qual pode ser baseado em uma narrativa.

A história dos videogames começou em 1958, quando o físico americano Willy Higinbotham, membro do projeto Manhattan, criou um simples jogo de tênis em seu osciloscópio, para entreter os visitantes de seu laboratório. Em 1968, o alemão Ralph Baer desenvolveu um sistema de jogos que poderia ser conectado a um aparelho de televisão, até então os jogos funcionavam em computadores, que eram geralmente restritos apenas a um seleto grupo de universitários.

A partir de década de 70 e 80, quando começaram a surgir os fliperamas e consoles de videogame que utilizavam controles e renderizavam gráficos, começou a popularização dos jogos eletrônicos ao público geral. Os primeiros jogos a surgir durante essa época foram Spacewars! e Pong. Ambos do gênero arcade, esses jogos se tornaram muito populares, sendo reconhecidos até o dia de hoje. Com o passar dos anos mais e mais videogames foram sendo feitos e cada vez se aproveitando mais da tecnologia disponível em suas épocas. Exemplos atuais como The Last of Us e Uncharted 4, que venderam respectivamente 1,3 e 2,7 milhões de cópias na primeira semana de venda (TASSI, 2013), mostram que não só os jogos ficaram altamente tecnológicos com gráficos impressionantes, mas que também possuem um mercado muito forte.

2.1.3 Jogos Arcade

Os arcades eram máquinas operadas por moedas ou fichas, compostas por uma tela e um conjunto de botões, geralmente localizada em espaços públicos como shoppings ou bares. Os jogos disponíveis em máquinas arcade tinham algumas características em comum: tinham controles simples mas que requeriam bastante precisão e coordenação motora, com uma dificuldade que crescia rapidamente, sendo muito complicados em estágios mais avançados. Além disso, o jogador podia jogar até perder a partida, sendo assim obrigado a comprar outra ficha.

A estratégia desse tipo de jogo para lucrar era balancear bem sua dificuldade: deveria ser difícil o suficiente para o jogador perder com rapidez, sendo obrigado a comprar várias fichas, mas não tanto a ponto dele desistir e se desestimular com o jogo.

Esses jogos apresentavam fases curtas e bem definidas, quando o jogador perdia durante uma fase ele retornava do início da mesma ou do último ponto de salvamento, mas isso não era um problema sendo parte da experiência falhar várias vezes antes de ser bem sucedido. Atualmente essas características podem ser encontradas em jogos de diversos

estilos sendo os de plataforma os mais recorrentes. Esse trabalho se propõem a ter como foco jogos com essas características.

2.2 QUALIDADE

Apesar de jogos eletrônicos, em sua maioria, terem como objetivo principal o entretenimento, sua qualidade é tão importante quanto a de um software tradicional, e seus usuários são tão críticos quanto. Essa seção tem como objetivo apresentar a importância da qualidade em jogos e definir conceitos importantes para o entendimento das características de um bom jogo.

2.2.1 A indústria

A indústria de jogos eletrônicos cresce a cada ano: de acordo com a SuperData, ela gerou uma receita de 104,6 bilhões de dólares em 2017 (alcançando pela primeira vez a marca de 100 bilhões), com grande parcela dessa receita vinda de jogos de celular, computador e console, respectivamente (CHAUX, 2017). Consequentemente, o dinheiro investido na produção de jogos também cresce: o jogo *Grand Theft Auto V* teve um investimento de 256 milhões de dólares, sendo muito até para os padrões de *Hollywood* (na época de lançamento do jogo o único filme que o ultrapassava em verba foi Piratas do Caribe: No Fim do Mundo) (CASSEY, 2015). Com investimentos tão grandes, é esperado um retorno à altura. Quando esse retorno não é alcançado as empresas podem enfrentar muitos problemas.

Existem alguns casos de fracassos famosos na indústria dos videogames. Um deles é o do jogo *E.T. the Extra-Terrestrial*, baseado no filme homônimo. Por se basear em um filme de tremendo sucesso comercial, o jogo foi licenciado para o console Atari 2600 por 21 milhões de dólares e seus produtores acharam que seu produto iria pelo mesmo caminho de sucesso do filme. As negociações acabaram em cima da hora, e o produto teve um prazo de cinco semanas e meia para ser concluído (STILPHEN,).

O resultado final foi tão ruim e o jogo foi criticado tão fortemente (de forma negativa) em todos os seus aspectos, que é considerado até hoje um dos piores jogos do mundo. Inclusive um documentário foi produzido para contar a história desse fracasso (PENN, 2014). Todo esse prejuízo e infâmia se deu pela falta de qualidade do jogo, no fato dele ter sido produzido às pressas e às cegas, sem contato direto com seu usuário final.

2.2.2 Qualidade em Software

A qualidade de software pode depender do tipo de aplicação que se espera de determinado sistema. O sistema de um aparelho hospitalar que mantém um paciente vivo, pela sua criticidade, deve ser à prova de falhas (pois a vida de uma pessoa depende disso), mas não precisa necessariamente ser de fácil utilização, pois seus usuários serão treinados para

Figura 1 – O jogo *E.T. the Extra-Terrestrial*

isso. Já um software de ensino para crianças, por exemplo, deve ser de fácil utilização devido ao seu propósito educativo, e eventuais falhas não causarão nenhum dano grave.

Por causa dessas diferenças, a ISO 9126 (1991) descreve uma padronização para que seja possível desenvolver software de qualidade, independente do seu propósito. A ISO 9126 define qualidade de software da seguinte maneira:

> Qualidade é a totalidade de características e critérios de um produto ou serviço que exercem suas habilidades para satisfazer às necessidades declaradas ou envolvidas

A fim de garantir essa padronização, a ISO 25010 (2011), sucessora da ISO 9126, definiu um conjunto de requisitos necessários que um software precisa, denominado *Systems and software Quality Requirements and Evaluation* (SQuaRE).

Esse conjunto de requisitos é divido em duas categorias: Qualidade de Produto e Qualidade em Uso, cada uma contendo diversas características.

A categoria Qualidade de Produto se refere à propriedades do *software* propriamente dito e é dividido em oito características principais: aptidão funcional, eficiência de performance, compatibilidade, usabilidade, confiança, segurança, manutenibilidade e portabilidade. Essas características principais são alto nível, mas a ISO detalha tambem subcaracterísticas de cada uma delas (figura 2).

A categoria de Qualidade em Uso foca na consequência da interação humana com o *software* e possui cinco características principais: eficácia, eficiência, satisfação, liberdade de risco e cobertura de contexto. Assim como na categoria citada anteriormente, também possui subcaracterísticas: satisfação possui utilidade, confiança e prazer; liberdade de risco possui mitigação de risco econômico, mitigação de saúde e risco de segurança e mitigação

Figura 2 – Critérios de qualidade definidos pelo SQuaRE

de riscos ambientais; e por fim, cobertura de contexto é dividido por completude de contexto e flexibilidade.

Logo, essa categoria de qualidade em uso enfoca o usuário e seus sentimentos quando utilizando um *software*, e é esse prisma que será explorado a seguir.

2.2.3 Qualidade em Jogos

Jogos eletrônicos também são softwares, mas não necessariamente a aplicação dos atributos definidos pela padronização internacional são suficientes para desenvolver um jogo de qualidade.

De acordo com Fullerton existem cinco fatores no desenvolvimento de um jogo que devem ser atingidos a fim de garantir sua qualidade (FULLERTON, 2008).

O fator Funcional indica se as funcionalidade do jogo permitem que ele seja jogável e se estão funcionando corretamente. O jogo é Internamente Completo se todos as ramificações e regras dele estão funcionando e sendo respeitadas. É Balanceado se sua dificuldade é ideal, não é muito fácil mas ainda assim desafiador. O jogo deve ser Acessível, significando que precisa ser de fácil entendimento, com boa curva de aprendizado e intuitivo. E por fim, o jogo deve ser Divertido, o que significa que deve engajar o jogador, entreter, desafiar e fazer com que ele deseje continuar jogando.

Alguns desses fatores, como o Funcional e Internamente Completo podem ser parci-

almente testados de forma parecida com os softwares tradicionais: testes automatizados que testam todas as execuções de cada funcionalidade e cenários.

Já os outros requisitos (Balanceamento, Acessibilidade e Diversão) são muito mais subjetivos, e requerem um contato maior com o usuário final a fim de coletar *feedback*. Fases de *beta testing* são comuns na indústria, para colocar o produto em contato com seu público alvo o quanto antes, e entender as suas necessidades.

É nessa fase de aprendizado, do jogador entrando em contato com o produto e experimentando com ele, que se tiram as primeiras conclusão sobre a direção que o desenvolvimento deve tomar. Existem diferentes processos para capturar essas métricas necessárias para um desenvolvimento com qualidade. Será detalhado a seguir um desses processos, chamado de Telemetria.

2.3 TELEMETRIA

Nesta seção, serão apresentados conceitos básicos para o entendimento do que é telemetria e como ela pode ajudar na área de desenvolvimento de jogos.

2.3.1 Desenvolvimento orientado a dados

A coleta de dados do uso de um software que podem ser mensurados e interpretados é muito importante a fim de ajudar o desenvolvedor a entender melhor alguma característica do seu produto e tomar ações a partir dessas informações para melhorá-lo.

2.3.1.1 Síntese

A síntese de uma característica de uso de um software é a composição de vários dados para a geração de uma informação. É possível, então, entender o comportamento médio do seu usuário, a fim de entender suas necessidades.

Um exemplo do uso de síntese no entendimento de um software é o gráfico de DAU (*Daily Active Users*, em português Usuários Ativos por Dia). Essa informação é útil principalmente para desenvolvedores de jogos *multiplayer* entenderem se o mercado está respondendo positivamente ou não ao seu jogo.

2.3.1.2 Análise

A análise, por outro lado, ao invés de unir todos os dados, permite o estudo de cada parte de forma independente e bem detalhada. É possível então entender o comportamento de um usuário específico, a fim de focar melhor em um nicho de usuários ou encontrar anomalias no software. Alguns casos que estariam fora de uma curva mediana na síntese, ainda assim podem ser interessantes e conseguiriam ser avaliados durante a análise.

Um exemplo de uso é analisar o comportamento de um jogador em algum ponto específico do jogo. Ao lutar contra um chefe, que decisões o jogador toma ou que estratégia ele usa? O que levou o jogador a seguir por um caminho ou outro? A resposta para essas perguntas podem ser obtidas através da análise do rastreamento do jogador e informar se o combate está muito difícil ou muito fácil, por exemplo.

2.3.2 Telemetria

Telemetria (do grego *tele* significa remoto e *metron* medida) refere-se à captação dados que permita uma medição a distância. Apesar do termo ser originalmente utilizado para transmissão de dados sem fio, como ondas de rádio ou sistemas infravermelhos, pode também se referir a comunicações via telefone ou rede de computadores.

É utilizada para monitoramento em diversos segmentos diferentes. Balões meteorológicos, por exemplo, capturam dados referentes a temperatura, pressão atmosférica e umidade e os transmitem para estações via radiossondas, onde eles serão processados e analisados.

Na área de softwares esta técnica pode ajudar, entre outras coisas, na detecção de falhas, no monitoramento de performance e na descoberta de quais funcionalidades os usuários usam mais.

No contexto de jogos eletrônicos, existem diversas áreas de atuação para telemetria como na análise de servidores de jogos com quantidade massiva de usuários, no entendimento dos dispositivos móveis no qual o jogo está sendo executado, na identificação do impacto de uma mudança no jogo (como uma nova arma ou novo cenário) e no monitoramento das interações entre usuários.

2.3.3 Telemetria no mercado de jogos

Na prática, o processo de telemetria pode ser usado para atuar na solução de uma ampla gama de problemas. A empresa *RAD Game Tools* possui uma ferramenta chamada *Telemetry* que se propõe a ajudar o desenvolvedor a otimizar e entender a performance da sua aplicação, mostrando gráficos com visualização do estado da aplicação e recursos de hardware consumidos, por exemplo. Essa ferramenta foi utilizada em grandes títulos AAA da indústria dos jogos, como *League of Legends* e *Grand Theft Auto V*.

Outra ferramenta interessante é o *Analytics*, disponibilizado como uma ferramenta complementar para a *Unity*. Um caso de sucesso da ferramenta foi com o jogo RollerCoaster Tycoon Touch com o objetivo de melhorar a performance em todos os dispositivos possíveis, eles usaram uma funcionalidade da ferramenta possibilita alterações dinâmicas nas configurações para uma melhor otimização que por meio da coleta de dados. Com isso chegaram a resultados bem interessantes como uma diminuição em 35% do *churn* (taxa de evasão) (UNITY,).

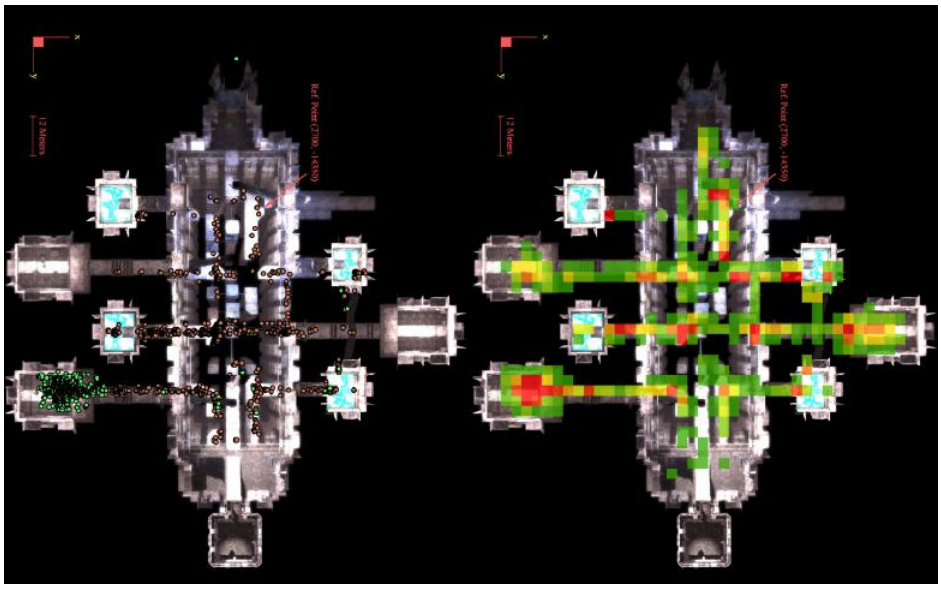

Figura 3 – Localização do jogador ao pedir dicas para solução do quebra-cabeça em *Tomb Raider: Underworld* (esquerda). Mapa de calor com escala do verde claro (poucos pedidos) para vermelho escuro (muitos pedidos). (DRACHEN; CANOSSA, 2013)

Entretanto, além da parte técnica de otimização de performance, é muito importante o uso da telemetria para entender o comportamento dos jogadores. É possível estudar a atuação dos jogadores de diversas maneiras, como por exemplo convidar grupos de pessoas para *play testing* e observá-las durante as partidas ou ter uma equipe de QA *in-house*, que vai jogar e reportar o *feedback* da experiência. Porém, a vantagem da telemetria, por capturar dados à distância, é de conseguir capturar dados referentes a um grande volume de jogadores (ou até todos), que estão jogando de sua própria residência (sem nenhuma mudança de comportamento, que pode ocorrer quando estão sendo observados enquanto jogam) o que traz resultados muito mais precisos do que utilizando grupos amostrais de *play testing*, por exemplo.

A *IO Interactive* utilizou análise espacial para estudar o comportamento dos jogadores em dois grandes títulos da empresa (DRACHEN; CANOSSA, 2013). Em *Kane and Lynch 2: Dog Days* foi avaliado como o jogador navega pelos ambientes, com foco principal em detectar pontos problemáticos (dificuldade de progressão muito alta ou existência de algum *bug*) na interação do jogador com suas armas. Já em *Tomb Raider: Underworld*, um jogo altamente focado em desafios e quebra-cabeças espaciais (por ser um *platformer* 3D), foram registradas todas as vezes que o jogador pediu dicas para solucionar os quebra-cabeças e sua posição no mapa do jogo ao requisitá-las (figura 3).

Essas métricas ajudaram os game designers a entender a dificuldade do desafio e como balanceá-lo.

A BioWare utilizou técnicas de captação de métricas para verificar com que atividades

Figura 4 – Processo de captação de métricas e aplicação do aprendizado, descrito por Phillip Derosa, diretor de *QA* da Bioware

os jogadores estavam gastando seu tempo no jogo Mass Effect e como balancear o uso de poderes especiais (DEROSA, 2007). A figura 4 descreve o *pipeline* utilizado pela empresa para iterar em seus jogos, com base em análises de como seus produtos estavam sendo jogados.

Além de grandes companhias, a telemetria pode ser principalmente útil em casos de jogos pequenos, para uma equipe limitada de funcionários e recursos, que não podem custear ferramentas utilizadas por grandes companhias. Chris Pruett cita como o simples ato de plotar no mapa do seu jogo os locais onde os jogadores estavam perdendo, ajudaram a detectar sérios problemas no design do seu jogo (PRUETT, 2010).

2.3.4 Proveniência de Dados em Jogos

Proveniência (*Provenance*), no contexto da arte ou bibliotecas digitais, se refere a história documentada de um objeto de arte ou a documentação dos processos no ciclo de vida de um objeto digital. O *Open Provenance Model* (OPM) (AL, 2010) é uma modelagem de proveniência que define claramente relações de causalidade entre processos, agentes e artefatos, que são os nós da relação.

Kohwalter et. al adaptaram o modelo *OPM* para o contexto de jogos eletrônicos (KOHWALTER; CLUA; MURTA, 2012), que serão brevemente descritos a seguir.

2.3.4.1 Artefato

Um artefato é definido como uma unidade imutável de estado, que pode ser representado como um objeto ou uma representação digital em um sistema. No universo dos jogos, o artefato vai ser qualquer item que pode ser utilizado pelo jogador ou que tenha algum contexto relevante para a narrativa, como por exemplo: armas, peças de roupa, livros, poções, etc.

2.3.4.2 Agentes

Agentes são entidades que atuam como catalisadoras de processos, e que podem facilitar, controlar ou afetar de alguma maneira sua execução. No contexto de jogos, agentes podem ser representados por personagens não jogáveis, como monstros, vendedores e aliados ou até mesmo por outros jogadores, em caso de jogos multiplayer.

2.3.4.3 Processos

Por fim, processos são ações ou sequência de ações executadas em artefatos ou causada por eles. Em jogos, processos podem ser ações ou eventos gerados por entidades inteligentes.

2.3.4.4 Relações de causalidade

O modelo OPM, de acordo com MOREAU et al. (AL, 2010), especifica as possíveis relações entre os tipos de nó previamente detalhados.

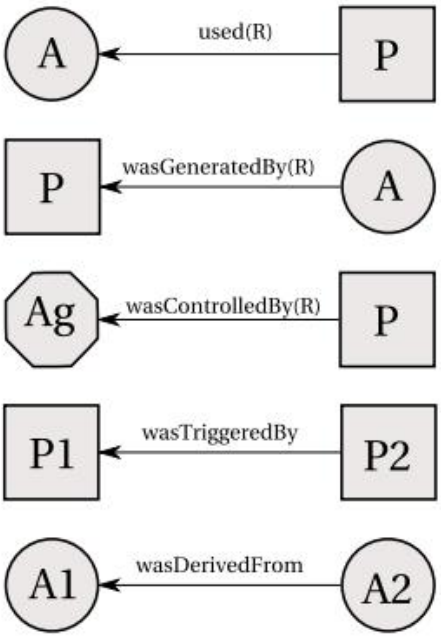

Figura 5 – Relações de causalidade entre nós, definidas em *The Open Provenance Model Core Specification (v1.1)* (AL, 2010)

Relação Causal: Representada através de um arco que demonstra a relação de dependência causal entre origem (efeito) e destino (fonte).

Artefato usado por Processo: Uma aresta de processo (P) para artefato (A), indica que este processo necessita da disponibilidade deste artefato a fim de concluir sua execução. Quando diversos artefatos estão conectados a um processo, isso significa que todos precisam estar disponíveis.

Artefatos gerados por Processos: Uma aresta de artefato (A) para processo (P), indica que é preciso que o processo inicie sua execução, a fim de gerar o artefato. Quando vários artefatos estão ligados ao mesmo processo, quer dizer que o processo precisa iniciar para que todos os artefatos sejam gerados.

Processo controlado por Agente: Uma aresta que liga um processo (P) a um agente (Ag) indica uma relação de dependência onde o agente controlou o início e fim do processo P.

Processos disparados por Processos: uma aresta que liga o processo P2 ao processo P1, é uma relação causal de dependência indicando que o início da execução do processo P1 é necessário para o processo P2 ser concluído.

Artefato derivado de Artefato: Uma aresta de um artefato A2 para o artefato A1 é uma relação que indica que o artefato A1 deveria existir para o A2 ser gerado. O estado de A2 depende da presença de A1.

3 TECNOLOGIAS

Essa seção tem como objetivo introduzir as ferramentas utilizadas para a realização do projeto.

3.1 C#

C# é uma linguagem de programação multi-paradigma (apesar de ser mais comumente utilizada com o paradigma de orientação a objetos) desenvolvida por volta dos anos 2000 pela Microsoft. A linguagem tem utilização ampla, mas brilha na construção de aplicações desktops voltadas para plataformas Windows.

Apesar de não ser uma linguagem nova, ainda está entre as 10 linguagens mais adoradas entre os desenvolvedores, como relatado em pesquisa realizada em 2018 (STACK..., 2018).

3.2 UNITY

Um motor de jogos (*game engine*) tem como função abstrair vários aspectos do desenvolvimento de um jogo, como renderização, física, sistema de colisão, input e outros, de modo que o designer ou desenvolvedor possam focar exclusivamente na lógica do seu jogo e em suas características únicas.

Figura 6 – Tela do editor da Unity

Unity é um motor de desenvolvimento de jogos com suporte multiplataforma lançado em 2005 durante a Apple Inc.'s Worldwide Developers Conference (WWDC). Atualmente seu editor é compatível com Windows, macOS e Linux (ainda em fase experimental) e têm capacidade de gerar jogos para 27 plataformas diferentes, desde PlayStation 4 até Samgung Smart TV.

A engine usa C# como sua linguagem de scripting principal, já tendo oferecido suporte à JavaScript e Boo, ambas descontinuadas na versão atual.

A ferramenta está ganhando muita popularidade entre desenvolvedores de jogos independentes (FAMULARO, 2018) por ser leve (em comparação com outras opções no mercado), possuir opção de plano gratuita e uma curva de aprendizado suave.

3.3 FIREBASE REAL TIME DATABASE

Firebase é uma plataforma da google que provê alguns serviços para abstrair programação server side, isto é, um *Backend-as-a-Service* (BaaS). Dentre os serviços oferecidos encontram-se banco de dados, autenticação, analytics, entre outros.

Para este projeto foi utilizado o Real Time Database, que através de uma API REST ou bibliotecas de integração, permite armazenar dados no formato JSON em um banco de dados hospedado na nuvem da Google. Essas dados são sincronizados em tempo real para todos os usuários conectados ao serviço. O serviço oferece integrações com algumas linguagens/tecnologias específicas, como iOS, Android e Unity, que foi a utilizada neste projeto.

3.4 JSON.NET

A fim de armazenar os dados no Firebase Real Time Database, é preciso primeiro convertê-los para o formato JSON. Para isso utilizamos o framework Json.NET da Newtonsoft.

A ferramenta open source permite a serialização e deserialização de qualquer objeto .NET com alta performance, e uma sintaxe de fácil leitura. Isto torna possível a conversão dos objetos que descrevem os eventos para objetos JSON, para serem salvos no Firebase.

4 PROPOSIÇÕES

Essa seção tem como objetivo explicar o modelo de telemetria proposto detalhando os conceitos envolvidos e explicar de forma conceitual o objetivo da ferramenta como um todo e de suas partes individuais.

4.1 COMPATIBILIDADE

É importante destacar que tipo de jogo é compatível com a ferramenta desenvolvida. Atualmente é possível utilizá-la em jogos com o formato similar ao de jogos *arcade* (detalhados previamente) no seguinte aspecto: fases/segmentos bem definidos (se contrapondo a jogos de "mundo aberto", onde o jogador tem liberdade de exploração)

4.2 OBJETIVO

O modelo de telemetria aqui proposto tem como objetivo principal uma visualização fácil e objetiva de ações tomadas pelos jogadores durante sessões de um jogo através da definição de eventos personalizados, dando poder de análise para o usuário da ferramenta. Permitindo assim interpretação das informações a fim de realizar qualquer tipo de ajuste necessário ou corroborar resultados de ajustes previamente realizados.

Apesar da capacidade analítica proveniente da ferramenta ser seu foco, ela também disponibiliza algumas informações sintetizadas sobre todos os jogadores, mostrando seu comportamento médio.

4.3 MODELO

4.3.1 Métricas disponíveis

4.3.1.1 Em síntese

Para detalhar as métricas de síntese disponíveis, é necessário primeiro definir os termos sessão e rodada.

Uma sessão representa o momento entre o jogador ter aberto o jogo pela primeira vez até fechá-lo. Incluindo o tempo gasto nos menus do jogo, não apenas o tempo efetivamente jogando.

Uma rodada é o momento entre o começo da fase, momento que o jogador começa a jogar efetivamente, e o fim da mesma (tendo sido concluída de forma vitoriosa ou não).

As métricas de síntese disponíveis são:

- Total de sessões

- Tempo médio de duração de uma sessão
- Quantidade média de rodadas por sessão

4.3.1.1.1 Total de sessões

Interessante para entender se existem muitas pessoas jogando. Junto de alguma ferramenta para guardar um histórico seria possível analisar o crescimento da popularidade do jogo.

4.3.1.1.2 Total médio de duração de sessões

Pode ser usado para analisar o engajamento dos jogadores, ou seja, o quanto o jogo consegue entreter.

4.3.1.1.3 Quantidade média de rodada por sessões

Dependendo do jogo o desenvolvedor pode querer que esse número seja maior ou menor. Uma maior quantidade pode ser um bom indicativo para jogos em que as fases são curtas e esperasse que o jogador re-jogue ela várias vezes. Uma menor quantidade pode ser um bom indicativo para jogos mais longos e com mapas maiores, onde esperasse que o jogador demore mais em uma fase antes de morrer.

4.3.1.2 Em análise

Existem jogos dos mais diversos tipos, então os dados necessários para avaliar o comportamento de um jogador podem variar drasticamente dependendo do gênero, plataforma e até público-alvo de um jogo. Por isso a ferramenta não disponibiliza métricas pré-definidas para análise.

Ao invés disso, propicia a criação de eventos customizados para cada tipo de jogo, a fim de rastrear a trajetória dos jogadores. O fato desses eventos serem genéricos e flexíveis, permitem que o desenvolvedor consiga utilizar a ferramenta independentemente do gênero ou características específicas do seu jogo.

4.3.2 Eventos

Neste modelo, um evento é um fato observável que acontece envolvendo uma ou mais entidades de um jogo. Os tipos de eventos, a forma como se relacionam com essas entidades e o mapeamento delas para o contexto de jogos foram inspirados no OPM (AL, 2010) e no trabalho de Kohwalter et al. (KOHWALTER; CLUA; MURTA, 2012), porém simplificados de forma a facilitar sua interpretação através de visualizações objetivas, feita de forma imediata pelo próprio editor da *Unity*.

Os dados do jogo são registrados em eventos, que podem ser disparados em momentos arbitrários, definidos pelo desenvolvedor. Além das informações pré-definidas atreladas a um evento, é possível atribuir a ele informações adicionais para um melhor entendimento em uma análise futura. Esses eventos podem representar qualquer interação do jogador, mudança no jogo ou ação de um NPC.

Existem três tipos de eventos definidos, que devem ser utilizados em diferentes situações, dependendo das entidades envolvidas, que serão detalhados a seguir.

4.3.2.1 Entidades

Foram mapeadas dois tipos de entidade passíveis de participação em um evento: agentes e objetos.

4.3.2.1.1 Agentes

Agentes são entidades que possuem inteligência, seja ela a de um jogador ou artificial. Qualquer ente que tenha vontade própria ou que possa tomar decisões, será considerado um agente. Alguns exemplos de entidades agente são: o jogador, um personagem que se movimente e converse com o jogador em um jogo de aventura e um carro adversário em uma corrida (que apesar de ser um objeto no consenso tradicional, em um jogo de corrida onde o piloto não aparece diretamente, esse torna-se a personificação do piloto).

4.3.2.1.2 Objetos

Objetos são utensílios ou instrumentos que não têm consciência no contexto do jogo. Não podem tomar decisões nem se mexer de forma independente. Alguns exemplos de objetos: Uma espada em um jogo de aventura, um carro em um jogo de exploração.

4.3.2.2 Evento Atômico

O evento atômico tem como propósito representar um acontecimento que não tenha consequência direta no estado do jogo, isto é, não influencia em nada o andamento de uma partida, do ponto de vista dos agentes presentes no jogo.

Esse tipo de evento é usado principalmente para acompanhar a posição de agentes ou objetos de interesse, como por exemplo a localização do jogador ou a posição de objetos posicionados proceduralmente (através de algoritmos) no cenário, que estão fora do controle direto do desenvolvedor e podem ser de interesse para uma futura análise (figura 8).

Figura 7 – Em *Paper Mario: Color Splash* um dos aliados de Mario, Huey, é um balde. Apesar de ser literalmente um objeto, ele é provido de inteligência e ajuda o personagem principal, por isso neste modelo é categorizado como um agente.

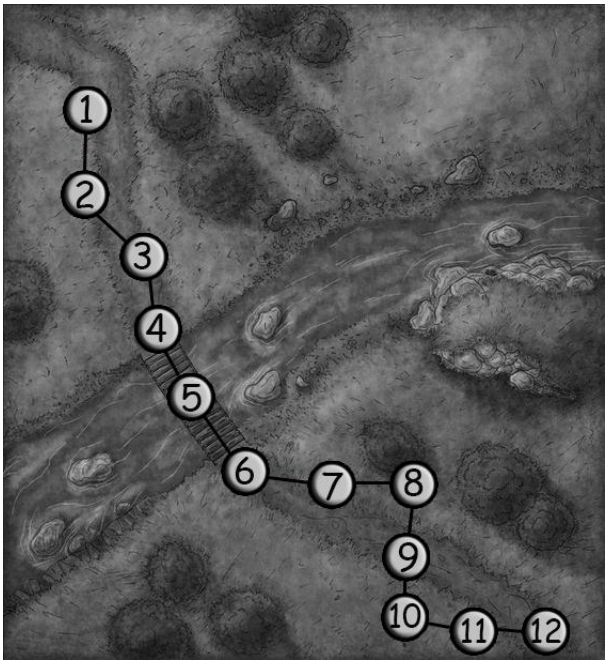

Figura 8 – Exemplo de uso de eventos atômicos para rastreamento do caminho feito pelo jogador. No exemplo, o jogador seguiu um caminho começando pelo nó 1, passando então por uma ponte, se aproximando de uma árvore no nó 8 e finalmente chegando até o nó 12.

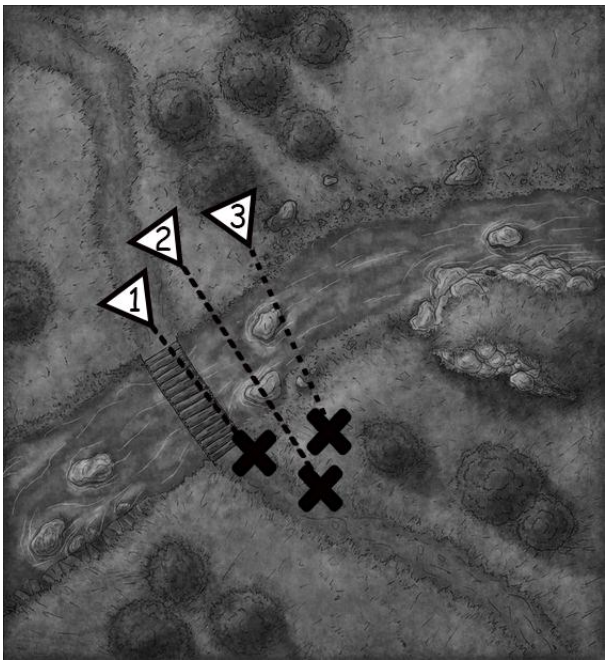

Figura 9 – Exemplo de uso de eventos em cadeia para registrar o lançamento de flechas e o momento em que elas atingem o alvo. No exemplo, cada triângulo numerado representa o momento em que uma flecha é disparada e estão ligados por uma linha pontilhada a um "X" que representa o momento em que a flecha atinge um alvo.

4.3.2.3 Evento em Cadeia

O Evento em Cadeia captura mudanças no estado do jogo, registrando o agente causador da mudança, e o agente ou objeto que a sofreu, conseguindo assim expressar relações de causa e consequência, para avaliar como decisões tomadas (por jogadores ou IAs) podem afetar o rumo de uma partida. Isso é possível mesmo quando a consequência não aconteça de forma imediata após sua execução.

Um cenário possível é um jogo onde o personagem use um arco e flechas para caçar. Atirar uma flecha é uma ação, então assim que ela for disparada, será registrado um evento na posição desse acontecimento de origem. Essa flecha irá percorrer sua trajetória, até atingir seu alvo ou algum obstáculo. De qualquer forma, quando a flecha atingir um agente/objeto, será registrado um evento de consequência, conectado ao que o originou. Dessa forma é possível avaliar, por exemplo, a precisão com que o jogador dispara suas flechas, quais tipos de presa ele atinge com mais frequência e quais são rápidas demais (figura 9).

Isso permite o ajuste desses valores (velocidade das presas ou velocidade da flecha) até se atingir um resultado balanceado, que resulte em uma experiência agradável ou desafiadora o suficiente, dependendo do propósito do jogo.

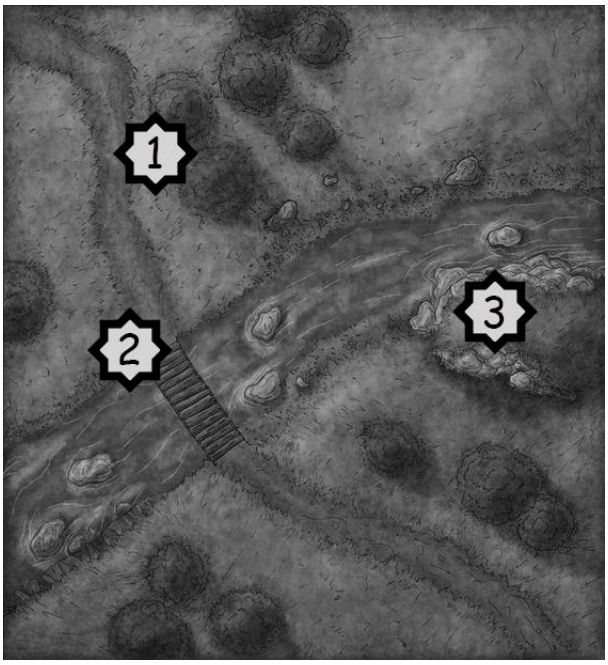

Figura 10 – Exemplo de uso de eventos únicos para registrar acontecimentos no jogo. No exemplo, o evento 1 seria do momento em que o jogador colhe uma planta, o evento 2 o momento em que o jogador aciona uma alavanca para descer a ponte levadiça e o evento 3 o momento em que o jogador interage com um rúnica que lhe restaura a energia.

4.3.2.4 Evento Único

Assim como o Evento em Cadeia, o Evento Único também causa mudanças no estado do jogo, porém essas mudanças não causam consequências em momentos posteriores. As consequências de um evento único são resolvidas de imediato no jogo.

Em um jogo, qualquer interação com objetos como uso de poções, ativação de diálogos ou recolhimento de itens (como pegar uma arma do chão) podem ser registrados como eventos únicos já que espera-se que esses eventos não irão gerar diretamente outros eventos (figura 10).

4.4 FERRAMENTA

A ferramenta funciona como uma extensão do motor de desenvolvimento de jogos *Unity*. Permite que o desenvolvedor tenha a capacidade de coletar informações remotamente sobre o comportamento dos jogadores durante a partida. Também possibilita a obtenção de métricas retiradas desses dados e a visualização do comportamento do jogador de forma fácil.

O registro se dá através da persistência das informações em um banco de dados, o que significa que várias instâncias do jogo terão seus eventos registrados de forma simultânea, aumentando o volume de dados e consequentemente amplificando o poder de análise do

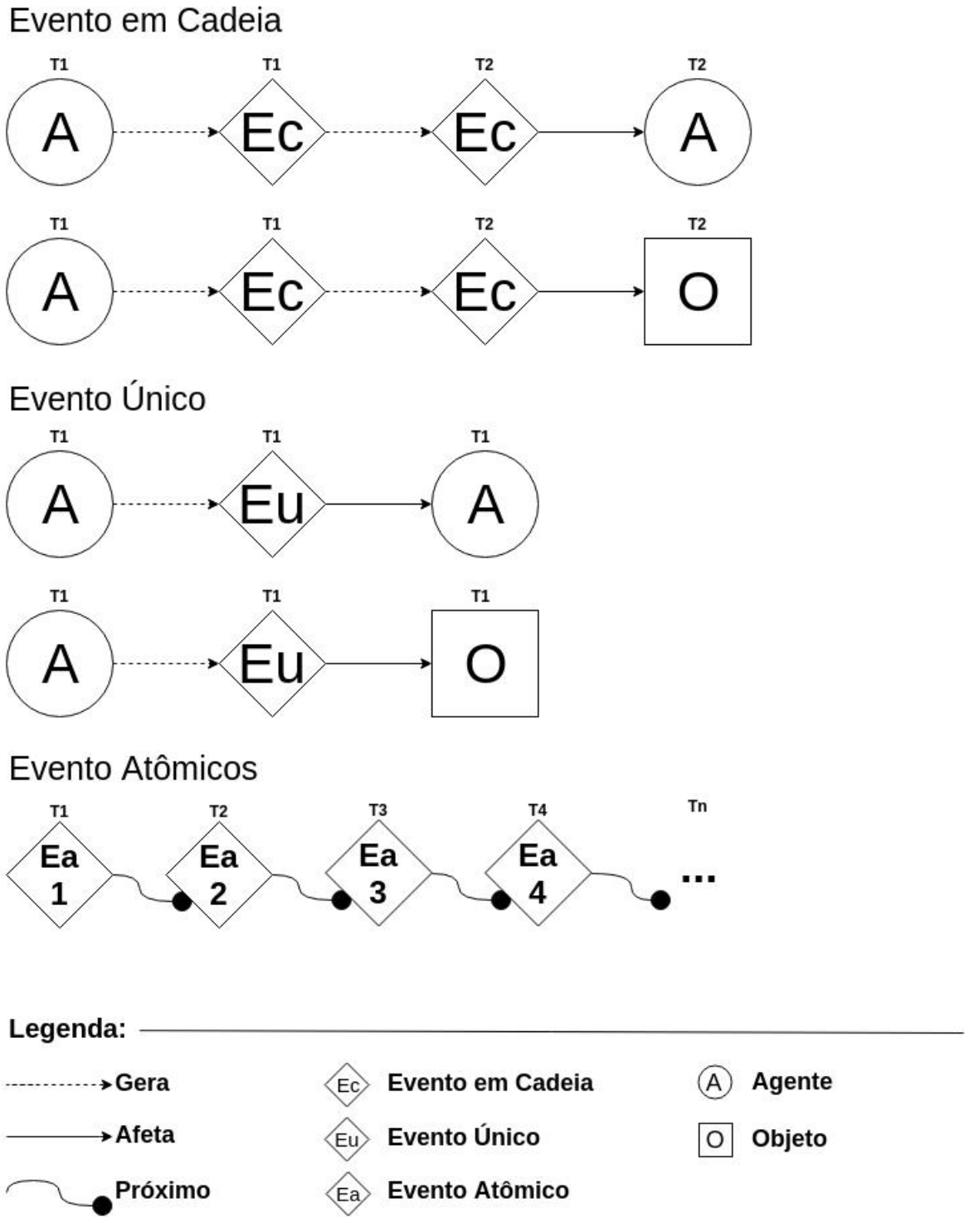

Figura 11 – Representação pictórica do modelo proposto

usuário.

Além disso, existe um conjunto de componentes responsável por controlar a captura desses eventos e visualizá-los de diferentes formas. Os componentes disponíveis são: *Telemetry Manager*; *Telemetry Synthesis*; *Telemetry Visualizer*; e por fim o *Node Info Window*. A seguir esses componentes e suas funcionalidades serão detalhados.

4.4.1 Componentes

4.4.1.1 Telemetry Manager

O componente *Telemetry Manager* (Gerenciador de Telemetria), é o âmago da parte de captura da ferramenta. Ele é responsável por ligar e desligar a captura de dados e informar quais cenas do jogo terão seus dados (eventos) registrados.

4.4.1.2 Telemetry Synthesis

Esse componente mostra as métricas de síntese dos dados capturados, como o total de sessões, média de duração de cada sessão e média de quantidade de partidas jogadas por sessão.

4.4.1.3 Telemetry Visualizer

É responsável por toda a parte de visualização no editor, de eventos capturados. Nesse componente o usuário define como quer representar visualmente cada um dos três tipos de evento (único, atômico e em cadeia) e qual partida ele deseja visualizar no momento. Após a renderização, é possível ver no editor, no mapa do jogo, todos os evento que ocorreram durante uma partida de forma precisa.

4.4.1.4 Node Info Window

Esse componente funciona em conjunto com o anterior. Enquanto o *Telemetry VIsualizer* renderiza cada evento na tela, o *Node Info Window* exibe detalhes desses eventos quando eles são selecionados no editor. Detalhes como posição, momento que ocorreu, nome e tipo de evento. Além disso, essas informações são personalizáveis via código, sendo possível adicionar qualquer tipo de informação extra nesses detalhes, no momento do registro do evento.

4.4.1.5 Mouse Handler

Esse é um componente obrigatório para o funcionamento do anterior (*Node Info Window*). Ele é responsável por rastrear a posição do mouse, e entender qual nó de evento o usuário está selecionando, permitindo que seus detalhes sejam expostos.

4.4.2 Conclusão

A ferramenta difere de algumas citadas anteriomente em alguns aspectos. Diferente da *Telemetry* (da RAD Tools), esta não auxilia em otimização e performance do jogo, focando unicamente em registrar e permitir a análise do comportamento dos jogadores. Já a ferramente PinGU, feita baseando-se no framework conceitual PinG (AL, 2017) se assemelha no aspecto de coletar métricas que auxiliem na análise de comportamento, mas visualiza os resultados de forma diferente: gera o grafo dos eventos em cima de uma imagem estática do mapa do jogo. Ao passo que esta ferramenta desenha esse grafo na própria cena do editor, dando flexibilidade de navegação especialmente útil em cenários 3D mais complexos, permitindo visualização detalhada de cada passo tomado.

Em suma, a ferramenta é capaz de registrar eventos customizados e exibi-los através de grafos no próprio cenário do jogo, através do editor da *Unity* e de forma sintetizada, para auxiliar no entendimento do comportamento dos jogadores.

5 MANUAL

Esta seção tem como objetivo explicar de maneira técnica como configurar e utilizar a ferramenta proposta, desde a implementação de eventos até a visualização das métricas.

5.1 CONFIGURANDO A PERSISTÊNCIA DE DADOS

Para que seja possível consultar dados observados durante partidas de um jogo, é preciso salvá-los em algum tipo de banco de dados. Por padrão, a ferramenta vem com o *Firebase Real Time Database* facilmente configurável, de forma que seja simples testá-la. Entretanto, caso necessário, é possível que o usuário configure sua própria solução de persistência.

5.1.1 Configurando o Firebase

Antes de começar a configuração, é necessário que o usuário tenha uma conta no serviço *Firebase* e um banco do tipo *Real Time Database* criado. Com esse banco criado, deve-se copiar sua URL de acesso (algo no formato https://nome-do-projeto.firebaseio.com) e colar esse valor na variável *dbUrl*, no arquivo *DBFirebase.cs*.

5.1.2 Configurando solução de persistência personalizada

Para possuir uma solução de persistência customizada, é necessário reescrever as funções de escrita e leitura no banco, que por padrão são implementadas apenas para o Firebase.

Portanto, basta o usuário acessar o arquivo *DBHandler.cs* da ferramenta e reescrever as funções *saveSessionsData* para escrita no banco e *loadSessionsData* para leitura, de acordo com a solução de persistência escolhida.

5.2 REGISTRANDO EVENTOS

A seguir será detalhado como criar cada tipo específico de evento e suas particularidades.

5.2.1 Eventos Atômicos

Para criar um evento atômico, é preciso criar um nó do tipo *Atomic* e então registrá-lo. Um evento atômico tem 3 parâmetros obrigatórios: tipo de nó, nome do evento e posição onde o evento ocorreu.

Código 5.1 – Criação de um evento do tipo atômico

```
TelemetryNode playerPosition = new TelemetryNode(
        TelemetryNodeType.Atomic,   // Tipo de nó
        "Posição do Jogador",       // Nome
        transform.position          // Posição
);
```

informações extras sobre o evento também pode ser passado, caso haja necessidade de guardar metadados adicionais, atrelado a um evento atômico específico.

Código 5.2 – Criação de evento do tipo atômico com dados adicionais

```
dynamic additionalMetaData = new ExpandoObject();
additionalMetaData.occurenceDate = "25/12/1997"

TelemetryNode playerPosition = new TelemetryNode(
TelemetryNodeType.Atomic,
        "Player Position",      // Nome do evento
        transform.position,     // Posição de ocorrência
        additionalMetaData      // Objeto de metadados adicionais
);
```

Após criar o nó do evento, este deve ser registrado, chamando-se o método *addNode* com o nó criado anteriormente sendo passado como parâmetro. Depois dessa etapa, esse evento já estará registrado, e assim que a aplicação for terminada eles será persistido, junto com todos os outros eventos registrados na sessão.

Código 5.3 – Registrando evento atômico

```
TelemetryCore.addNode(playerPosition);
```

5.2.1.1 Referenciando um evento atômico em outro

Em alguns casos, o usuário pode querer referenciar eventos atômicos uns nos outros, para conseguir rastrear algum acontecimento durante o jogo. Um exemplo disso é quando é necessário rastrear a posição do jogador durante todo o curso da rodada, para saber exatamente por onde ele passou. Para isso basta criar vários eventos atômicos com um intervalo de tempo específico entre eles (definido pelo usuário da ferramenta) e conectá-los usando o método *setLink* de um nó.

Código 5.4 – Script de rastrear posição do jogador

```
public class PositionTracker : MonoBehaviour {
  [SerializeField] float trackingDelay = 0f;
  List<Vector3> positionsTracked = new List<Vector3>();
  int lastPositionId = -1;

  void Start () {
    InvokeRepeating("track", 0f, trackingDelay);
  }

  void track() {
    TelemetryNode playerPosition = new TelemetryNode(
          TelemetryNodeType.Atomic,
          "Player Position",
          transform.position
        );

    if(lastPositionId != -1) {
          playerPosition.setLink(lastPositionId);
    }

    lastPositionId = TelemetryCore.addNode(playerPosition);
  }
}
```

Por padrão, todo nó de evento criado possui um atributo chamado *link* com o valor -1, isto é, não possui ligação com nenhum outro nó. Usando o método *setLink*, é possível sobrescrever esse valor com o identificador do nó que queremos referenciar no evento.

No script de exemplo (código 5.4), a função *track* é chamada em intervalos regulares definidos pelo usuário (*trackingDelay*) e seu papel é criar um nó de posição do jogador e verificar se é a primeira vez que ele está sendo criado, caso não seja a primeira ocorrência, deve referenciar o evento anterior no atual, usando a função *setLink*.

Utilizando esse método, quando esse evento for renderizado em tela, todos os seus nós estarão conectados ao nó anterior, criando um rastro da posição do jogador.

5.2.2 Eventos em Cadeia

Como o evento em cadeia acontece em dois momentos diferentes, pois ele tem uma causa e uma consequência e pode ter um intervalo entre esses dois acontecimentos, a criação de um evento deste tipo é um pouco diferente da criação de um evento atômico.

Para exemplificar sua criação, será usado o seguinte cenário: o jogador dispara um projétil que vai atingir em algum momento seu adversário (ou algum outro obstáculo, caso ele erre o disparo).

Neste caso, primeiro é necessário criar o evento e salvar seu identificador, que será usado posteriormente.

Código 5.5 – Iniciando criação de evento em cadeia

```
shotEventID = Telemetry.createChainEvent ("Disparo do jogador",
    transform.position);
```

Deste modo, a variável *shotEventID* possui o identificador do evento "Disparo do jogador", então assim que o projétil atingir um adversário ou obstáculo, podemos criar o evento consequência desse disparo, passando como parâmetro esse identificador e finalizando esse evento em cadeia.

Código 5.6 – Finalizando criação de evento em cadeia

```
Telemetry.createChainEvent ("Inimigo atingido", transform.position,
    shotEventNodeId);
```

Esse evento de consequência pode ser criado independente do jogador ter acertado ou não seu alvo, dependendo das necessidades de análise do desenvolvedor. Ele pode ser lançado também caso o jogador erre ou atinge algum objeto de relevância que queira ser analisado, como por exemplo um barril explosivo ou inflamável, a fim de causar alguma explosão.

Cada chamada do método *createChainEvent*, já cria o nó e registra o evento.

Código 5.7 – Outros exemplos de como concluir um evento em cadeia

```
Telemetry.createChainEvent ("Barril explosivo atingido", transform.
    position, shotEventNodeId);

Telemetry.createChainEvent ("Disparo errado", transform.position,
    shotEventNodeId);
```

5.2.3 Eventos Únicos

O evento único é criado de forma semelhante ao evento em cadeia. A diferença principal é que ele não é dividido em dois momentos como o anterior. Então quando um evento acontecer, é preciso simplesmente chamar o método *createSingleEvent*, que ele será criado e registrado.

Código 5.8 – Criando um evento único

```
Telemetry.createSingleEvent ("Jogador perdeu", transform.position);
```

5.3 INTERFACE

É preciso introduzir alguns conceitos da *Unity*, já que este projeto de telemetria foi criado como um Pacote *Unity*, isto é, uma extensão para essa ferramenta.

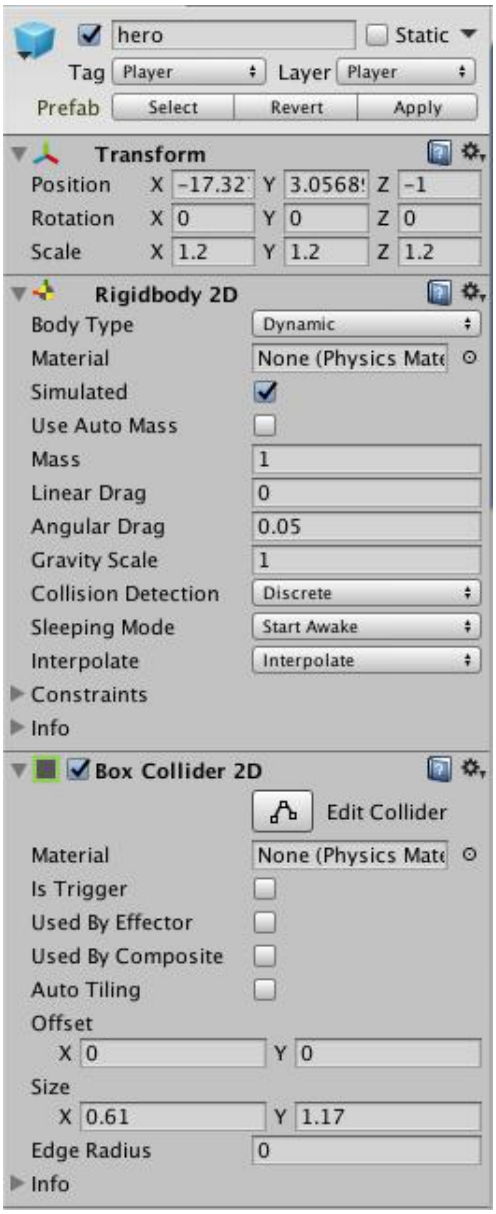

Figura 12 – Exemplo de uma entidade chamada "hero" utilizando 3 componentes nativos da *Unity*: *Transform*, *Ridigbody 2D* e *Box Collider 2D*

A *Unity* divide as diversas fases de um jogo usando um sistema de Cenas, definidas pelo desenvolvedor. Pode-se imaginar uma cena como uma fase do jogo, uma cena contém todos as decorações, obstáculos, personagens e menus que compõe uma fase. A ferramenta de telemetria vai registrar eventos de cada uma dessas cenas, de forma separada e independente.

Na *Unity*, um Componente, é um *script* que define o comportamento das entidades presentes nas cenas. Cada entidade pode ter um ou mais componentes. A *Unity* já possui diversos componentes que definem diferentes comportamentos prontos para serem utilizados, como o componente *Transform* (que define a posição, rotação e tamanho de uma entidade), o *Rigidbody* (que define como uma entidade se comporta com o sistema

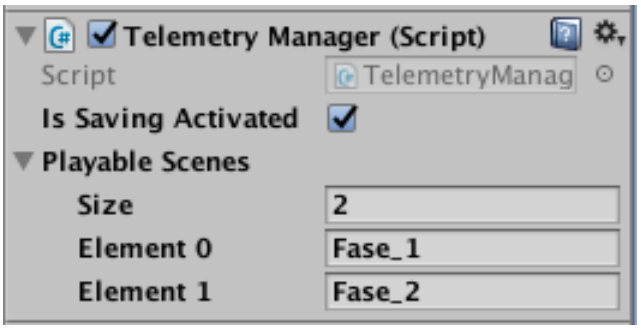

Figura 13 – Componente *Telemetry Manager*

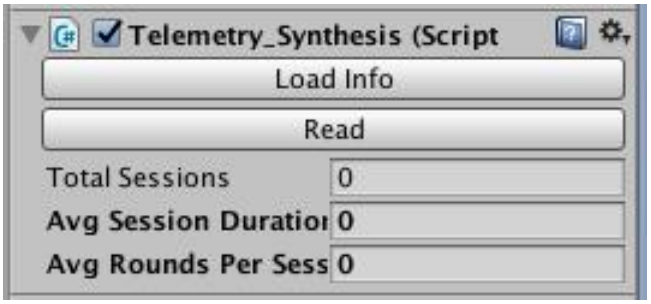

Figura 14 – Componente *Telemetry Synthesis*

de simulação de física do jogo), entre outros.

Na janela Hierarquia são listadas todas as entidades em uma determinada cena.

O pacote de telemetria é composto por diversos componentes, cada um com uma utilidade específica, que serão detalhados a seguir. É possível achar esses componentes ao olhar para a janela Inspetor após selecionar a entidade Telemetry. Essa entidade pode ser encontrada já definida na pasta da biblioteca.

5.3.1 Gerenciador de telemetria

No componente *Telemetry Manager* (figura 13), o campo *Playable Scenes* especifica em quais cenas o sistema de telemetria vai ser executado, isto é, quais cenas vão ter os eventos nela ocorridos registrados.

No campo *Is Saving Activated*, é possível ativar e desativar quando é desejado que os eventos de uma partida sejam salvos. Essa opção é útil, por exemplo, quando o desenvolvedor executar o jogo para testar funcionalidades, mas não quer que esses testes sujem o banco de dados.

5.3.2 Métricas de síntese

O componente *Telemetry Synthesis* (figura 14) agrupa as métricas de síntese descritas no capítulo anterior. O botão *Load Info* carrega os dados atualizados do banco para a *Unity* e o botão *Read* preenche os campos, calculando seus valores.

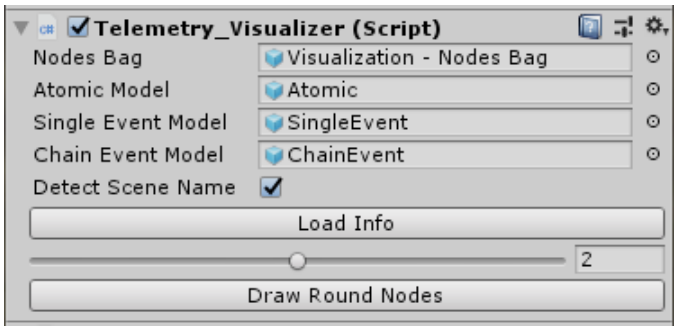

Figura 15 – Componente *Telemetry Visualizer*

5.3.3 Visualização de eventos

A visualização é a parte mais importante da ferramenta, sua utilização é dividida através de três componentes que devem estar na cena: o de renderização, o de informação de nós e configurador do *mouse*.

5.3.3.1 Renderizando e customizando a aparência de eventos

O componente *Telemetry Visualizer* (figura 15) é responsável por definir quais ícones vão representar cada um dos três tipos diferentes de evento, agrupar eventos gerados para não poluir a interface do editor e renderizar os eventos na tela.

Esse renderizador funciona da seguinte forma: para cada evento ocorrido nessa cena (que pode ser especificada manualmente ou deixar a própria ferramenta detectar qual a cena carregada no momento) será instanciado um ícone na cena com o modelo definido pelo usuário, na posição de ocorrência do evento.

Nos campos *Atomic Model*, *Single Event Model* e *Chain Event Model* o usuário pode definir qual ícone (modelo 3D ou *sprite* 2D) ele deseja utilizar para representar o evento visualmente na tela do editor.

O campo *Nodes Bag* referencia em que objeto serão guardados os eventos gerados. Como para cada evento é instanciado um ícone em cena, eles são agrupado para não poluir a tela de hierarquia.

O botão *Load Info* carrega e processa os dados do banco e o *Draw Round Nodes* renderiza na tela todos os eventos da cena carregada no momento (caso a *checkbox Detect Scene Name* esteja selecionada) ou de uma cena especificada pelo desenvolvedor. Ao carregar os dados, o *slider* mostra todas as partidas que ocorreram na cena, permitindo ao usuário escolher qual rodada ele deseja *plotar* no editor.

5.3.3.2 Detalhamento de eventos

Depois que todos os eventos de uma rodada especificada estiverem renderizados na tela do editor, é possível ver detalhes do evento ao passar o cursor sobre seu ícone. Para

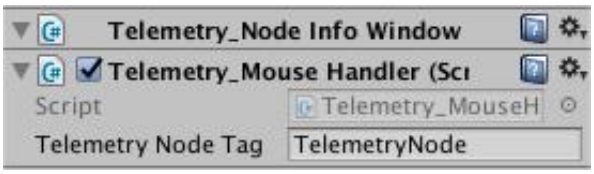

Figura 16 – Componentes *Node Info Window* (ainda vazio, devido a nenhum evento ter sido selecionado) e *Mouse Handler*.

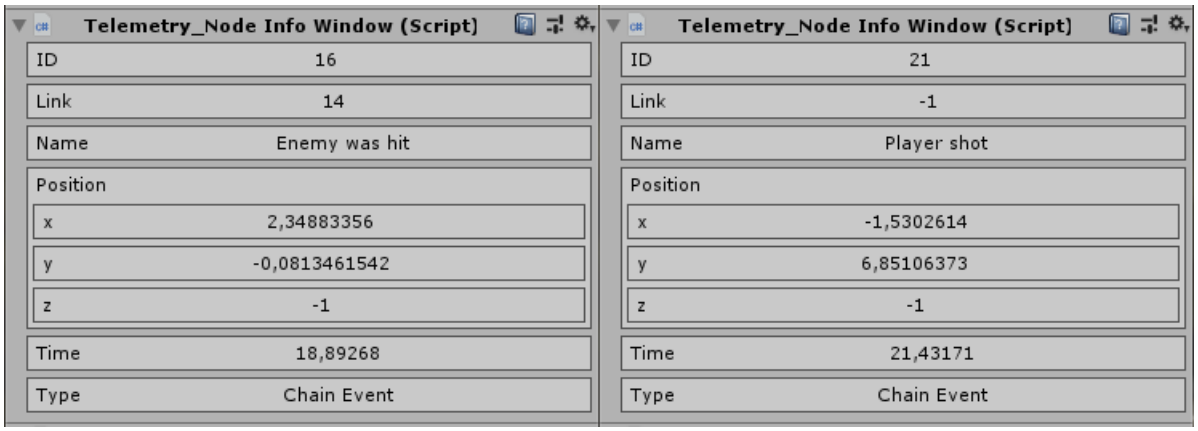

Figura 17 – Exemplo do componente *Node Info Window* mostrando detalhes quando dois eventos distintos são selecionados no editor.

ativar essa funcionalidade, dois componentes devem ser adicionados na cena: O *Node Info Window*, que mostra dos detalhes do evento e o *Mouse Handler*, que rastreia a posição do cursor para verificar qual evento está sendo selecionado (figura 16).

Além de informações padrão comum a todos os eventos, essa janela também mostra informação personalizada adicionada pelo usuário. A figura 18 exemplifica um componente mostrando detalhes de um evento atômico que registrou a posição do jogador durante o jogo. Na imagem é possível ver o campo *Info* que contém informações personalizadas para análise do desenvolvedor. Lá foi registrada a informação *looking* que informa se no momento do registro do evento o jogador estava olhando para a direita ou para a esquerda, em um jogo *sidescroller*

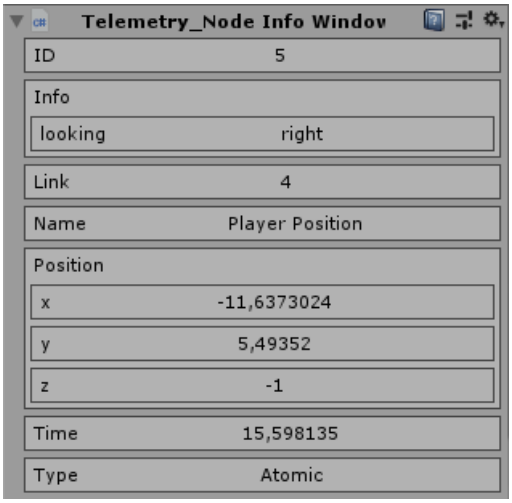

Figura 18 – Componente de detalhes do evento mostrando informações personalizadas.

6 EXEMPLOS DE APLICAÇÃO

Esta seção detalha o uso da ferramenta em dois jogos distintos, para testar sua aplicabilidade em diferentes cenários. Um dos jogos será *Heratoth*: um jogo autoral 3D, do gênero *stealth* e o outro um jogo amostra da *Unity* 2D, do gênero *Sidescroller Shooter*, que vem junto com a *engine* a fim de exemplificar seu uso.

6.1 HERATOTH

Heratoth é um jogo *stealth*, isto é, com mecânicas furtivas, onde o jogador deve passar despercebido por seus inimigos para chegar em determinado local enquanto coleta objetos de interesse.

6.1.1 Conceito do jogo

Em Heratoth, o jogador encarna um arqueólogo a procura de uma antiga relíquia. Em busca dessa relíquia, acaba preso em uma tumba, onde precisa encontrar a relíquia, desvendar os segredos da tumba e o mistério do sumiço de outros arqueólogos enquanto evita ser encontrado pelos seres que lá habitam.

As mecânicas do jogo são simples: o jogador não pode ser visto pelos seres da tumba (que têm seu alcance de visão claramente representado, para auxiliar o jogador) e deve encontrar a relíquia. Durante o percurso, o jogador pode ler diários de outros exploradores, que pereceram na mesma aventura.

O jogo possui dois finais. O final ruim ocorre quando o jogador encontra a relíquia sem antes ter lido todos os diários perdidos, portanto conseguindo a jóia sem entender completamente a história por trás dela e da tumba. O final bom ocorre quando o jogador encontra a relíquia depois de ter lido o relato de todos os outros exploradores através de seus diários.

O objetivo principal do uso da ferramenta de telemetria aqui é entender a dificuldade de não ser encontrado pelos inimigos e de encontrar todos os diários perdidos.

6.1.2 Eventos mapeados

Foram criados eventos para rastrear acontecimentos considerados importantes no jogo: movimentação do jogador, derrota, coleta de itens e fase concluída.

6.1.2.1 Eventos Atômicos

A movimentação do jogador foi registrada como uma sequência de eventos atômicos salvos com um intervalo de tempo curto entre eles (neste caso, um segundo). Assim é

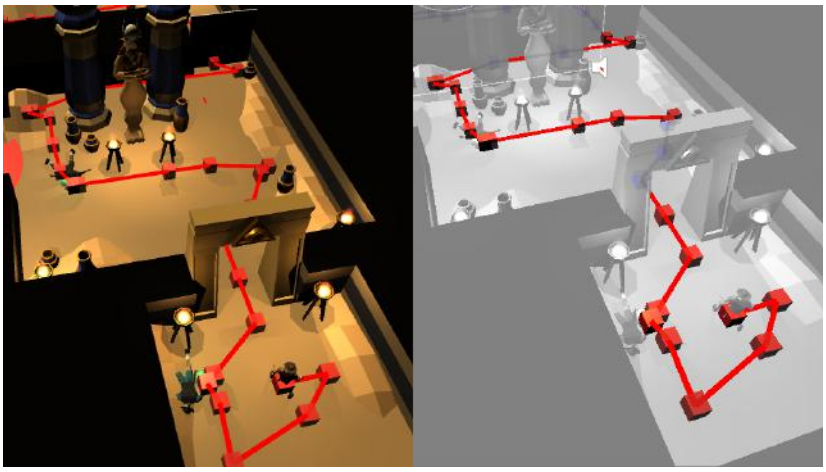

Figura 19 – Visualização dos eventos atômicos no jogo *Heratoth*: rastreamento da movimentação do jogador.

possível avaliar através do percurso do jogador a cobertura do mapa e com isso entender se existe um caminho preferido pelos jogadores diante de uma bifurcação ou se tem alguma região do mapa que não está sendo explorada (figura 19).

6.1.2.2 Eventos Únicos

Dois tipos de acontecimentos foram registrados como Eventos Únicos: a leitura dos diários e o término da fase (que é disparado com a interação do jogador com a relíquia).

Na leitura do diário, foi adicionada uma informação a mais (além das pré-definidas) no evento, que é o tempo de leitura do diário. É registrado quanto tempo o jogador deixa a janela com texto aberta (figura 21). A intenção por trás dessa métrica foi avaliar se os jogadores estavam realmente lendo o conteúdo dos diários ou interagindo com eles apenas para contabilizar o item como encontrado.

O evento de interação com a relíquia, que dispara o término da fase, registra qual dos dois finais possíveis o jogador alcançou.

6.1.2.3 Eventos em Cadeia

O acontecimento do jogo mapeado neste evento foi a derrota do jogador, pois um jogador perde quando é avistado pelo seres que povoam a tumba. Neste evento, é possível saber a posição do jogador e do inimigo que o avistou. Essa visualização possibilita a análise da dificuldade do caminho que o jogador deve percorrer para concluir a fase.

A figura 22 mostra o momento que um jogador estava se dirigindo para o sárcofago para pegar a relíquia, quando foi avistado por um inimigo, perdendo a partida.

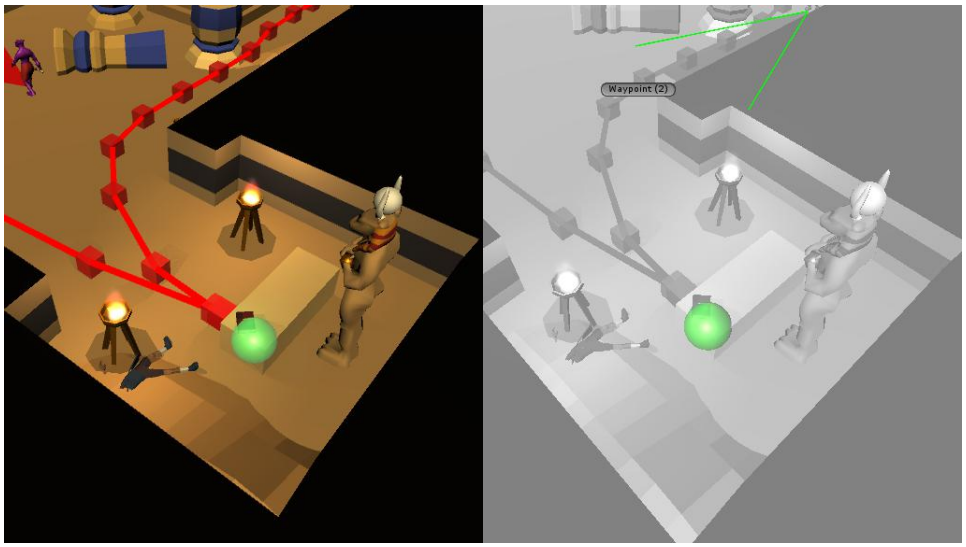

Figura 20 – Visualização de um Evento Único: momento que o usuário interage com o diário em cima do altar.

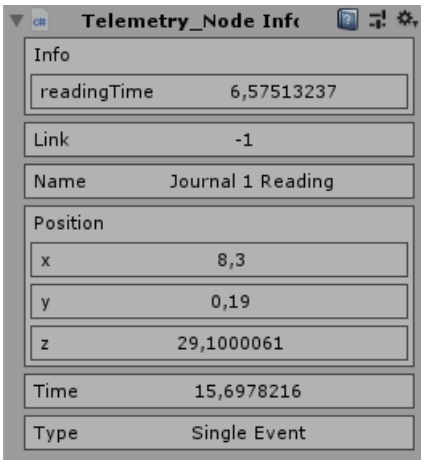

Figura 21 – Detalhes do evento de leitura de um diário. O campo *Time* registra quanto tempo o jogador deixou a tela com texto do diário aberta.

6.1.3 Conclusões

Com esses eventos mapeados e algumas partidas registradas, foi possível entender alguns pontos de melhoria no jogo, a fim de garantir uma melhor experiência para os jogadores.

6.1.3.1 Trecho difícil

Um trecho da tumba se mostrou difícil de ser ultrapassado: um corredor estreito, escuro e comprido, com um inimigo fazendo uma vigia constantemente no local. A maioria dos jogadores perdeu ao percorrer o corredor, pois não é possível avistar o inimigo antes de entrar, e ele se locomovia rápido demais. Sua velocidade mais o fato de ser um corredor estreito impossibilitavam o jogador de escapar.

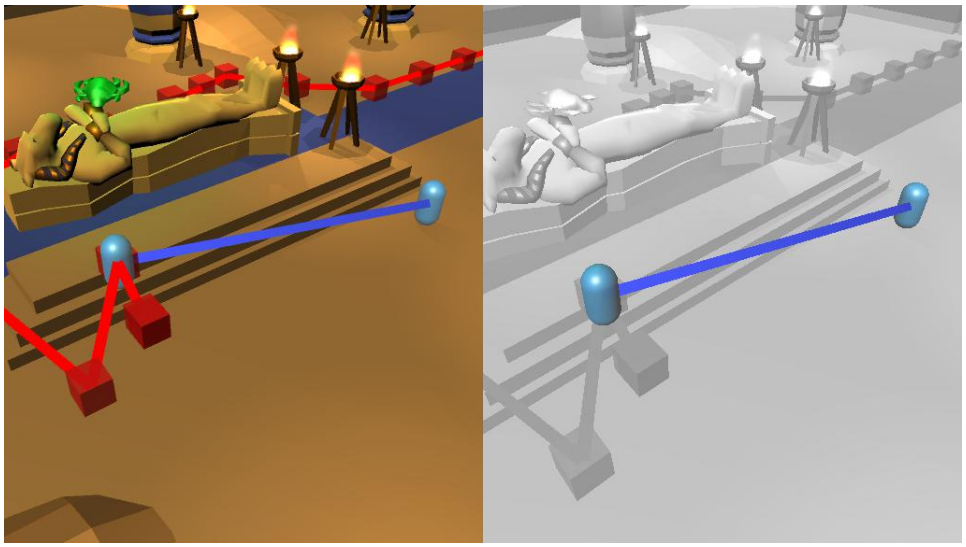

Figura 22 – Visualização de um Evento em Cadeia: momento que o usuário é encontrado por um inimigo no jogo e suas respectivas posições.

A solução para minimizar esse problema foi diminuir a velocidade do inimigo e adicionar uma sala anexa ao corredor, onde o jogador pode se esconder e esperar que a ameaça passe, antes de prosseguir.

6.1.3.2 Encontrar todos os diários

A ideia principal era ter dois finais, dependendo da quantidade de diários que fossem encontrados durante o jogo, mas a quantidade de jogadores encontrando todos os diários foi muito pequena. Foi discutido o motivo disso acontecer: a localização do último diário fica além da relíquia (que dispara o fim do jogo) e em momento algum do jogo é listado a quantidade total de diários. Dessa forma, nada motiva o jogador a procurar por todos.

Uma possível melhoria para esse caso seria um indicador durante a partida de quantos diários existem na fase, e quantos o jogador já encontrou, incentivando uma exploração mais diligente de sua parte.

6.2 JOGO EXEMPLO DA UNITY

É um jogo *2D*, com visão lateral do gênero arcade, inspirado no *Super Crate Box* desenvolvido pela holandesa *Vlambeer*. Este jogo é um projeto pronto, criado pelos desenvolvedores da própria *Unity*, que tem seu código disponibilizado gratuitamente, com a finalidade de aprendizado, sendo um jogo tutorial. Foi criado para mostrar as novas funcionalidades de desenvolvimento *2D* adicionadas na versão 4.3 da ferramenta.

Figura 23 – Captura de tela de uma partida do jogo exemplo da Unity, utilizado como um dos exemplos de uso da ferramenta de telemetria

6.2.1 Conceito do jogo

Nesse jogo você controla o que parece ser uma batata de bigode que tem uma bazuca e precisa matar os alienígenas que estão invadindo Londres.

As mecânicas são: andar para a direita e para esquerda, pular, atirar com sua bazuca, pegar caixas que caem do céu com granadas, que podem então ser arremessadas. Ao encostar em um alienígena o jogador perde vida, até chegar a zero, que é quando o jogador perde. Se cair para fora do cenário, também perde.

6.2.2 Eventos mapeados

Foram criados eventos para rastrear acontecimentos considerados importantes no jogo: movimentação do jogador, disparos de bazuca e o momento que o jogador pega uma caixa com granadas que cai do céu.

6.2.2.1 Eventos Atômicos

Com este evento foi mapeado o movimento do jogador. Pode-se observar na figura XYZ que o mesmo realizou vários pulos, aqui vistos pelos arcos no caminho traçado (figura 24).

6.2.2.2 Eventos Únicos

Esse evento foi utilizado para mapear quando e onde o jogador pagou a caixa com uma bomba que cai do céu. A imagem XYZ mostra a visualização da localização da caixa quando foi pega pelo jogador.

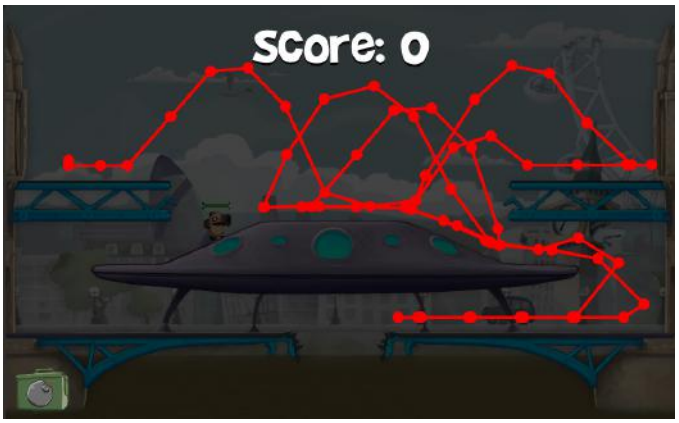

Figura 24 – Visualização dos eventos atômicos: rastro da movimentação do jogador durante a partida.

Figura 25 – Visualização dos eventos únicos: localização onde o jogador pega as caixas.

6.2.2.3 Eventos em Cadeia

Este evento mapeia o momento que o jogador dispara o tiro de bazuca e o momento que esse tiro atinge algo. É possível distinguir esses dois momentos passando o mouse em cima dos triângulos (figura 26) e observando seus nomes no componente *Telemetry Node Info*. O disparo (causa) tem o nome “Player shot” e o momento que atinge algum objeto (consequência) gera evento relativo ao alvo atingido.

Caso o jogador acerte um alienígena, o evento consequência será "Enemy was hit", caso acerte um caixote será "BombPickup was hit"e no caso de acertar algo irrelevante para as mecânicas do jogo, como partes do cenário, por exemplo, o evento "Something was hit"será gerado. Esse último evento (quando o usuário acerta objetos irrelevantes) poderia ter sido omitido, mas foi adicionado apenas para verificar a precisão da mira do jogador.

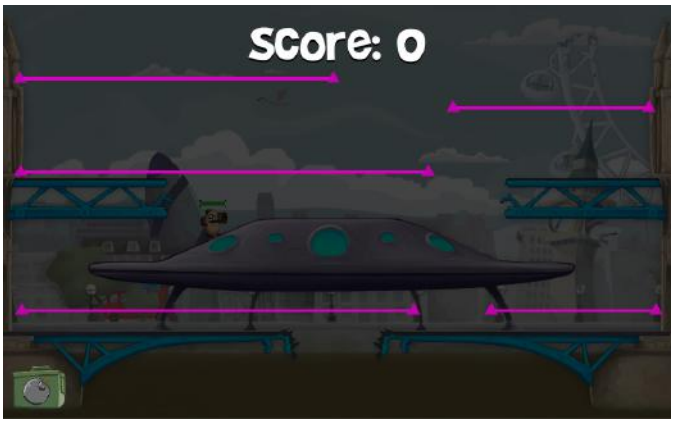

Figura 26 – Visualização dos eventos em cadeia: momento de disparo e momento de acerto.

Telemetry_Node Info Window	
ID	41
Link	36
Name	Something was hit
Position	
x	-20,0740166
y	-6,67897844
z	-1
Time	23,2862949
Type	Chain Event

Telemetry_Node Info Window	
ID	36
Link	-1
Name	Player shot
Position	
x	4,92598248
y	-6,67897844
z	-1
Time	22,2763863
Type	Chain Event

Figura 27 – Detalhes sobre os eventos ao passar o cursor por cima dele. O campo *Link* do evento de ID 41, referencia o evento de ID 36.

6.2.3 Conclusões

Apesar dos dados recolhidos terem sido úteis na análise da perfomance dos jogadores, a abordagem dessa seção será diferente da anterior, onde os eventos registrados foram analisados unicamente a fim de melhorar o produto (o jogo *Heratoth).* Nesse caso, por não se tratar de um jogo autoral, será feita uma auto-avaliação da própria ferramenta de telemetria proposta, tentando encontrar pontos de melhoria.

A ferramenta funcionou bem em um jogo 2D porém notou-se que neste jogo, como o cenário é pequeno e espera-se que o jogador o percorra inteiro várias vezes durante uma partida, a visualização dos eventos acabou sendo prejudicada por haver muitos deles sobrepostos (figura 28).

Uma ideia para melhorar essa visualização no futuro é ter um controle para selecionar intervalos de tempo nos quais eventos serão mostrados. Por exemplo, em uma partida que durou cinco minutos, o desenvolvedor poderia selecionar um intervalo de tempo de trinta segundos e ir avançando esses intervalos para ver eventos que aconteceram depois.

Assim visando diminuir a sobreposição de eventos e permitir que o desenvolvedor consiga efetivamente acompanhar o que aconteceu no decorrer da partida.

Figura 28 – Visualização de todos os eventos de uma partida

7 CONCLUSÃO

7.1 CONSIDERAÇÕES FINAIS

O processo de telemetria é muito útil para desenvolvedores medirem a qualidade de seus jogos e ajustá-los para que proporcionem a melhor experiência possível para o seu público. Muitos estúdios utilizam ferramentas rebuscadas, complexas e de alto custo como solução. Porém, como relatado, para pequenos desenvolvedores, soluções simples já são suficientes para agregar alto valor de análise.

Neste trabalho foi criada uma ferramenta de visualização simples de compreensão e utilização, *open source* (para extensibilidade colaborativa) e vinculada a *Unity*, uma ferramenta muito conceituada e utilizada no meio de desenvolvedores de jogos.

Através do disparo de eventos de três tipos (propostos no modelo), é possível mapear todo tipo de acontecimento em um jogo, independente de decisões estéticas ou técnicas, sendo necessário apenas uma estrutura compatível com o funcionamento da ferramenta (jogos com estrutura similar a de jogos *arcade*). Essa flexibilidade permite que a jogabilidade de uma gama enorme de jogos seja analisada através da visualização dos eventos, gerada pela ferramenta.

7.2 TRABALHOS FUTUROS

Algumas funcionalidades podem ser adicionadas a ferramenta para expandir sua compatibilidade em jogos com dinâmicas diferentes da proposta neste trabalho e aperfeiçoar a visualização dos eventos.

Pelo fato de registrar dados de fases (cenas, na *Unity*) individualmente, a ferramenta se encaixa bem em jogos com dinâmica similar à de jogos arcade, porém para jogos com fases longas e contínuas ou totalmente sem o conceito de fase, não é muito bem aplicado. Isso porque se o usuário salvar seu progresso e desligar o jogo para continuar posteriormente, quando ele retornar, todos os dados serão salvos em um novo registro, sem se relacionar com o seu progresso anterior.

Outro aspecto é referente à visualização para jogos onde o jogador fica toda a duração da partida no mesmo cenário, como o exemplo do jogo da *Unity*, citado na seção 6.2, onde a visualização pode ficar muito poluída devido ao alto número de eventos registrados. Uma solução para esse problema seria, dados todos os eventos de uma partida, filtrá-los pelo tempo de ocorrência. Assim seria possível ver os eventos aos poucos, mas ainda na ordem de acontecimento (como uma linha do tempo dos eventos), para manter a qualidade analítica.

REFERÊNCIAS

AL, L. M. et. The open provenance model core specification (v1.1). **Future Generation Computer Systems**, jun 2010.

AL, T. C. K. et. Capturing game telemetry with provenance. **SBGames**, nov 2017.

AVEDON, E. M.; SUTTON-SMITH, B. **The Study of Games**. [S.l.]: Irish Press, 2015.

CASSEY, D. Grand theft auto v is the most expensive video game in history at 265m. **CarThrottle**, 2015.

CHAUX, G. **Trends and insights on games and interactive media**. [S.l.], 2017.

DEROSA, P. Tracking player feedback to improve game design. **Gamasutra**, 2007.

DRACHEN, A.; CANOSSA, A. Evaluating motion: spatial user behaviour in virtual environments. **International Journal of Arts and Technology**, 2013.

ESPOSITO, N. A short and simple definition of what a videogame is. **University of Technology of Compiègne**, New York, 2005.

FAMULARO, J. What indie developers think of unity in 2018. **PC Gamer**, mar 2018.

FULLERTON, T. **Game Design Workshop: A Playcentric Approach to Creating Innovative Games**. 2. ed. [S.l.]: Morgan Kaufmann, 2008.

HUIZINGA, J. **Homo Ludens: O Jogo Como Elemento da Cultura**. 1. ed. [S.l.]: Perspectiva, 2017.

KOHWALTER, T. C.; CLUA, E. W. G.; MURTA, L. G. P. Provenance in games. In: **SBGames**. [S.l.: s.n.], 2012.

PENN, Z. **Atari: Game Over**. 2014. Filme.

PRUETT, C. Hot failure: Tuning gameplay with simple player metrics. **Game Developer Magazine**, sep 2010.

SALEN, K.; ZIMMERMAN, E. **Rules of Play: Game Design Fundamentals**. [S.l.]: MIT Press, 2003.

STACK Overflow Developer Survey. 2018.

STILPHEN, S. **DP Interviews Howard Scott Warshaw**. Interview.

TASSI, P. 'the last of us' has the biggest launch of the year with 1.3m sold. **Forbes**, 2013.

UNITY. **Roller Coaster Tycoon hits new heights: How Unity's LiveTune feature helped Nvizzio increase long-term retention by 50%**.