Murphy: Um Estudo de Caso utilizando Unity para Desenvolvimento de Jogos Casuais Mobile

Wanessa de Caldas Teotônio
Centro de Pesquisa, Desenvolvimento e Inovação Dell Dell LEAD
Fortaleza, Brasil
wanessacaldast@gmail.com

Allysson Allex Araújo
Gr. de Est. em Sistemas de Informação e Inovação Digital Universidade Federal do Ceará (UFC)
Crateús, Brasil
allysson.araujo@crateus.ufc.br

Pamella Soares
Prog. de Pós-graduação em Ciência da Computação Universidade Estadual do Ceará (UECE)
Fortaleza, Brasil
pamella.soares@aluno.uece.br

Jerffeson Souza
Prog. de Pós-graduação em Ciência da Computação Universidade Estadual do Ceará (UECE)
Fortaleza, Brasil
jerffeson.souza@uece.br

Matheus Paixão
Lab. de Ciência dos Dados e Inteligência Artificial Universidade de Fortaleza (Unifor)
Fortaleza, Brasil
matheus.paixao@unifor.br

***Abstract*—Casual mobile games represent an essential segment of the digital games market due to their accessibility, public reach, and democratization of the development process. However, creating a digital game is complex and multidisciplinary, where challenges commonly emerge during the development process. Thus, sharing and discussing the insights and lessons learned from projects under development with the community becomes relevant and insightful. Through a case study, this paper reports the development process of Murphy, a casual, mobile, and roguelike game, built with the Unity platform. In addition to providing rich detail on the ideation, storytelling and level design stages, we contribute towards a technical contextualization on the adoption of specific technologies, especially Unity, that we expect to be useful for other casual game developers and researchers.**

Keywords*—*Casual Game Development, Unity, Mobile Gaming, Development Process

I. Introdução

O segmento *mobile* contempla uma grande parcela do mercado de jogos digitais, em especial os desenvolvidos para o sistema operacional Android. Segundo a Pesquisa Game Brasil [1], realizada em 2020, 86,7% dos brasileiros utilizam o celular para jogar. Entre eles, 81,4% são usuários Android e 13,3% do seu concorrente mais próximo, o iOS. Uma amostra representativa desse mercado é consumido por jogadores casuais (67,5%), isto é, preferem jogos de acesso rápido, aptos a serem jogados em períodos curtos, que não demandam de muito conhecimento e que não necessitam de dedicação regular. Sob o ponto de vista de desenvolvimento de projetos, tal categoria de jogos demonstra-se promissora em decorrência do custo de produção reduzido quando comparado à produção de jogos para consoles.

Diante desse cenário, destaca-se a adoção de diferentes *game engines*, sobretudo a Unity, a qual corresponde a 50% de participação no mercado e conta atualmente com 100 milhões de jogadores, sendo, inclusive, utilizada em 54% dos 1000 jogos *mobile* com mais receita arrecadada [2]. A Unity permite gráficos e mecânicas que suportam diferentes tipos de efeitos, iluminação, textura e elementos físicos.

Visando agregar à consolidação das pesquisas na área, justifica-se a relevância em se discutir, junto a comunidade, o processo de desenvolvimento, incluindo avanços, desafios e lições aprendidas. Macedo e Rodrigues [3], por exemplo, apresentam o FunCopter, um jogo casual para plataformas móveis baseado em Unity. Além do jogo, os autores discutem aspectos relacionados ao processo de desenvolvimento, bem como limitações sob o ponto de vista gráfico. Em especial, o uso do Unity contribuiu para a produtividade devido ao editor visual e capacidade de *export* para diferentes plataformas. Por sua vez, Chaves e Ávila [4] relatam o *design* do Blind Runner, um jogo casual de corrida infinita, concebido como *audiogame* e construído com Unity. Ambos fazem parte de um escopo construtivo, ou seja, relatam a experiência de desenvolvimento de um jogo casual.

Assim, o objetivo principal deste artigo consiste, através de um estudo de caso, relatar o processo de construção de um jogo casual, em progresso, para Android utilizando Unity. Tal jogo, denominado Murphy, contempla características *roguelike*, *sidescrolling* e *point-and-click*. Como contribuições, destacam-se: i) amplo detalhamento do processo de ideação do jogo, incluindo o *level design* e *storytelling* (diferentemente de [3] e [4]); ii) um relato técnico sobre a adoção do Unity

em conjunto com tecnologias *open source* e iii) uma discussão das lições aprendidas.

O trabalho está dividido em 5 seções, além desta introdução. Na Seção II, detalha-se o *game design* e sua ideação. Na Seção III, discute-se a implementação do projeto. Na Seção IV, apresentam-se as lições aprendidas e na Seção V evidenciam-se as considerações finais.

II. Game Design Document (GDD)

Esta seção apresenta os detalhes e a descrição dos principais elementos que compõem o projeto Murphy, incluindo *storytelling*, personagens e *level design*.

A. Visão Geral

O jogo apresenta a estória de um jovem, chamado Murphy, que, após atingir a pontuação máxima em um *quiz* sobre astronomia e engenharia espacial, recebe a oportunidade de se aventurar como um astronauta. O jovem torna-se o piloto de testes de uma agência espacial, a Nebulosa Aloysius, que realiza pesquisas sobre transporte aeroespacial em cápsulas quânticas gravitacionais, uma nova tecnologia de transporte desenvolvida pela agência. O objetivo do jogo é guiar Murphy, localizado na Terra, até à nave espacial.

O nome Murphy, designado para o jogo e para o personagem principal, é uma homenagem ao engenheiro aeroespacial da Força Aérea Americana Edward Aloysius Murphy. A ideia foi inspirada pela popular lei de Murphy, que diz "Se alguma coisa pode dar errado, dará". Essa "lei" é folcloricamente empregada como uma justificativa bem (ou mal) humorada e pessimista, alegando que se alguma coisa deu errado foi porque já estava destinada a dar errado. Essa lei surgiu em 1949, quando Murphy participava de um experimento que testava efeitos da aceleração e desaceleração em pilotos de aeronaves. A experiência envolvia um conjunto de dezesseis medidores de aceleração colocados em diferentes partes do corpo humano. Neste episódio em particular, o técnico instalou todos os dezesseis medidores da maneira errada. Diante disso, Murphy pronunciou a icônica frase.

A viagem do personagem, da Terra até à nave, é feita através de trilhos coloridos, que iniciam no chão e terminam quando Murphy aterriza na nave. O personagem viaja em uma cadeira, presa a esses trilhos através de uma força magnética, que permite que ele vá subindo até que atinja a altitude que a nave se encontra. Os trilhos são construídos pelo próprio jogador no decorrer do nível. O desafio é que, para surgirem novos trilhos, o jogador precisa acertar os alvos localizados no lado oposto, como mostra a Fig. 1. A cor do trilho determina a cor do tiro que será disparado e a cor do alvo que deverá ser atingido. Cada tiro que acerta o alvo corretamente faz com que um novo trilho apareça e o jogador suba de altitude, aproximando-se do objetivo. A cor desse novo trilho é gerada de forma aleatória. Cada tiro que erra o alvo faz com que o jogador perca dois trilhos. O jogo acaba quando o jogador não possui mais trilhos na tela e cai, pois são estes que permitem sua sustentação, ou ele cumpre o objetivo final. Existem, no máximo, três trilhos na tela do jogo. Portanto, se o jogador errar uma vez ele precisa, obrigatoriamente, acertar os alvos seguintes para construir novos trilhos. Se errar duas vezes seguidas, ele perderá esse nível, perderá a pontuação conquistada e deverá começar novamente, até concluir.

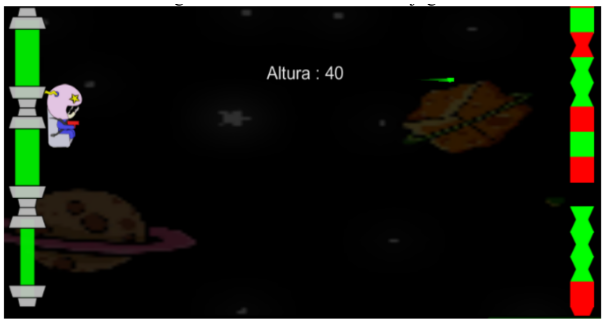

Fig. 1. Cena de um nível de jogo.

Essa característica é comum em jogos do tipo *roguelike*, os quais possuem, basicamente, duas características principais: cenários aleatórios e o fato de que se o jogador falhar em algum ponto antes de concluir a fase, ele irá começá-la novamente. Os cenários gerados aleatoriamente têm como propósito fazer com que o jogo se torne algo imprevisível. O fato de que, ao morrer, o jogador voltará ao início da fase, pode parecer negativo pois seria mais vantajoso poder salvar o progresso. Porém isto permite manter um certo nível de dificuldade, fazendo com que o jogador fique atento, melhore suas habilidades e se supere a cada tentativa.

Enquanto esse elemento promove o constante desafio de superação das habilidades do jogador, o cenário composto de forma aleatória permite que a fase não fique repetitiva, garantindo que o jogador não perca o interesse. O jogo é dividido em níveis (fases) com dificuldades e desafios crescentes. O jogador precisa concluir um nível inteiro, sem perder todas as chances de acertos, para que um próximo nível seja desbloqueado e se aproxime do objetivo final.

Além do estilo *roguelike*, o Murphy também é um jogo *sidescrolling* e *point-and-click*. *Side-scrolling* é a expressão utilizada para jogos em que o jogador move o personagem de um lado da tela para outro, com a câmera se deslocando somente na lateral, de forma vertical ou horizontal [5]. No caso do Murphy, a rolagem acontece de forma vertical. O personagem começa embaixo, no chão, e a câmera o acompanha até o topo, simbolizando a ida da Terra até o espaço. A característica *point-and-click*, pela qual interações com objetos e personagens são acionadas tocar na tela [6], é utilizada para disparar os tiros que saem da arma do personagem e irão atingir o alvo. Quando o jogador clicar no espaço de tela onde a arma está localizada, ela efetuará o disparo. Caso acerte o alvo, o jogador poderá construir novos trilhos que farão o personagem subir ou perder os trilhos que possui, caso erre.

B. Elementos de Cena

A Fig. 2 representa o primeiro esboço de fase do jogo. A partir dela, elaborou-se os elementos de cena, com a utilização

da ferramenta livre GIMP[1]. Os elementos da fase são: i) os trilhos que dão sustentação ao personagem e o levam até a nave; ii) a cadeira onde o personagem está preso aos trilhos; iii) o personagem portando uma arma; e iv) os alvos que deverão ser atingidos pelos tiros disparados.

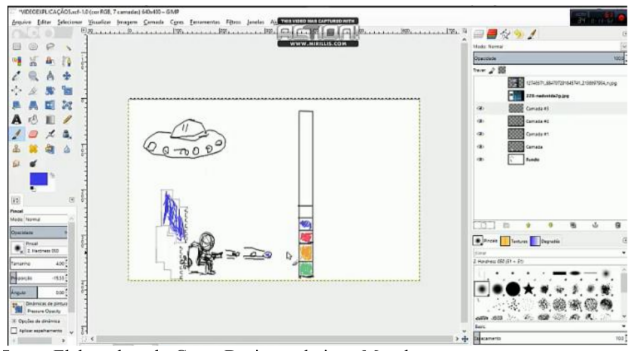

Fig. 2. Primeiro esboço da fase do jogo.

O cenário foi idealizado de forma linear, de baixo para cima, como se não existissem paradas durante o decorrer do jogo. Assim, tem-se o personagem viajando desde o chão e, à medida que os alvos são acertados, ele irá subindo e passando pelas diferentes camadas da atmosfera até o objetivo final, ou seja, a nave de destino. O cenário do jogo representa as camadas da atmosfera, sendo mostrado na Fig. 3. Entretanto, nos pontos de transição entre as fases, o jogador é levado às fases secundárias, com um novo cenário, onde ele deverá cumprir pequenos objetivos. A partir da pré-visualização dos elementos básicos que compõem o cenário, foi possível definir como seriam feitas as interações entre objetos e personagens durante o tempo de jogo.

C. Personagens

O jogo possui 5 personagens: Murphy, Rafael, Uba-Spock, Thaísa e Fish. Murphy é um jovem muito inteligente, apaixonado por astronomia. Ele foi contratado como piloto de teste de uma agência espacial, após obter a pontuação máxima em um jogo de perguntas e respostas. Para Murphy, realizar essa viagem é a concretização de um sonho alimentado desde a infância. Como denotado na criação do personagem (ver Fig. 4), em sua versão final, Murphy veste uma roupa especial, com as cores da agência que o contratou, a Nebulosa Aloysius. Murphy é o único personagem jogável e sua ação é atirar nos alvos com a pistola e aproximar-se cada vez mais da nave.

Rafael é um gênio na área da astrofísica, também trabalha na agência aeroespacial e gosta de cozinhar para os amigos. O Uba-Spock é um capitão de uma agência de pesquisa e proteção de seres galáticos de outro planeta. Ele pousou na agência Nebulosa Aluysius após sua nave ser atingida por um asteroide. Desde então ele pesquisa uma forma de consertá-la para voltar ao seu planeta. Thaísa é pesquisadora e proprietária da agência. Ela desenvolveu uma nova tecnologia para viagens aeroespaciais, com a ajuda de Rafael, e contratou o Murphy como piloto de testes. Fish é um gato que mora na agência e realiza algumas viagens com o Murphy. Esses personagens interagem com o Murphy em fases secundárias, cada um com objetivos distintos.

[1]Disponível em: https://www.gimp.org

D. Level Design

Existem dois tipos de fases: a viagem e as *quests*. A viagem é o nível principal, quando o personagem se desloca através dos trilhos para ir de um ponto a outro. Quanto a mecânica, o jogador deve acertar o alvo que possui a mesma cor do tiro, determinada pelos trilhos atrelados ao personagem. Ao acertar, surgirá um novo trilho que fará o personagem subir e chegar mais próximo ao destino.

Durante a viagem existem algumas paradas, para que o jogador possa salvar o progresso e executar as *quests*. As *quests* são fases secundárias e funcionam como pontos de transição entre uma parada e outra no nível principal. São importantes, pois permitem salvar o progresso e estabelecem uma pausa, onde o jogador pode descansar do modo rápido e restaurar a coordenação motora dos disparos. Nas fases secundárias, em uma mecânica *point-and-click*, o jogador explora a cena visando cumprir objetivos que correspondem ao toque correto em determinados elementos na tela como, por exemplo, entregar a comida do Fish aos personagens. Desse modo, em cada *quest* ele interage com um personagem *Non-player Character* (NPC) e desbloqueia novas trilhas sonoras. O jogo possui 11 níveis intercalados, sendo 6 no modo viagem e 5 *quests*, os quais representam o percurso até a nave de destino.

A pontuação é estabelecida de acordo com a quantidade de acertos e erros dos tiros nos alvos. Cada vez que o jogador erra um alvo, ele perde a pontuação acumulada equivalente a dois acertos, ou seja, mesmo que o jogador consiga concluir o nível, quanto mais ele errar, mais baixa será a pontuação. A pontuação será salva nos *checkpoints* existentes em cada término de uma *quest*. Caso o jogador falhe em um nível, toda a pontuação acumulada naquele nível é perdida e ele terá que recomeçar a partir do último ponto que foi salvo. Esse mecanismo de pontuação foi estabelecido para que o jogador sinta-se desafiado a conquistar a pontuação máxima. É um método usado para manter a atenção do jogador no jogo, fazendo com que ele repita o nível várias vezes, até que consiga superar sua pontuação ou de outros jogadores.

A primeira fase corresponde ao primeiro teste de viagem aeroespacial. Murphy iniciará no solo e chegará na agência Nebulosa Aloysius, localizada numa plataforma no céu. Ao chegar na plataforma, Murphy entra na agência e encontra Fish miando constantemente. Sua missão é encontrar comida e alimentar o gato. Ao completar a *quest*, o jogo será salvo. Após alimentar o gato, Murphy descobre que a agência está captando sinais de outro planeta, mas que não consegue identificar, pois é necessário uma maior altitude para melhorar a qualidade do sinal. Diante disso, Murphy partirá em uma nova viagem, que inicia na agência e termina no Centro Espacial de Pesquisas. Ao chegar no destino, o personagem entra no centro de

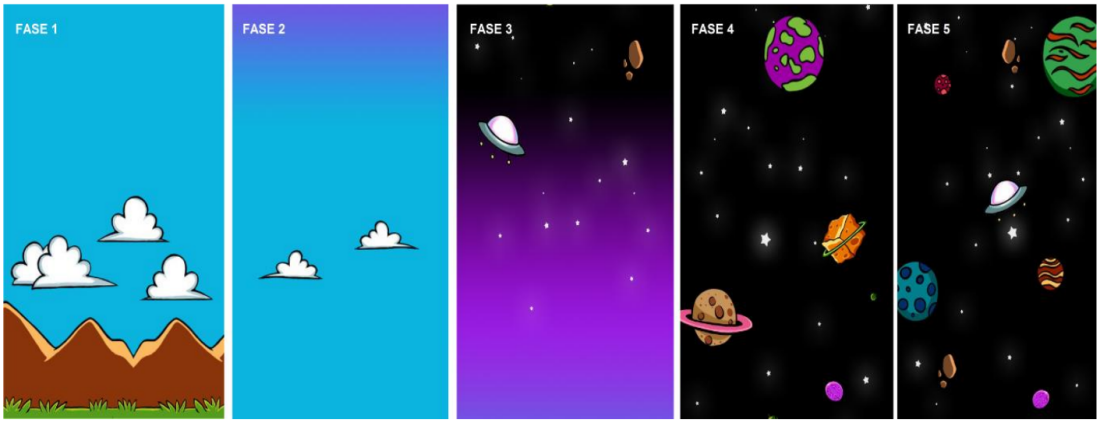

Fig. 3. Evolução de cenários do jogo Murphy.

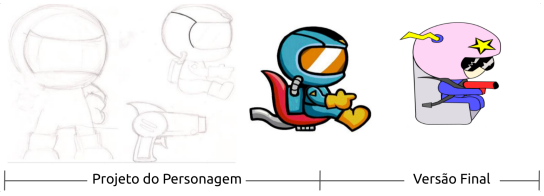

Fig. 4. Composição do personagem Murphy e sua versão final.

pesquisas e tem como *quest* gravar o sinal recebido e enviá-lo para Thaísa, que está na agência.

Enquanto Thaísa decifra a mensagem, Rafael avisa Murphy que ele precisará subir até o planeta mais próximo para buscar o ingrediente secreto da receita de macarronada espacial, que só é cultivado nesse planeta. Após o diálogo, o jogo será salvo e iniciará mais uma fase do modo viagem, que irá do centro de pesquisa até o próximo planeta. Ao chegar, Murphy colhe o ingrediente e o envia para a agência, dentro de uma cápsula quântica gravitacional, semelhante à que ele viaja. Após a cena de envio, Thaísa avisa a Murphy que decifrou a mensagem. Um extraterrestre, chamado Uba-Spock, do planeta Vulcanium, quer viajar até a Terra. Entretanto, um objeto não identificado impede a passagem de Uba-Spock. Murphy continua sua viagem até o objeto para verificar se é seguro que Uba-Spock trafegue por ele.

Ao se aproximar do objeto, Murphy percebe que é uma enorme quantidade de balões perdidos no espaço, utilizados anteriormente em um passeio mal sucedido. Como não há tripulantes, o personagem tem a *quest* de estourar os balões para desobstruir o caminho de Uba-Spock. O amável extraterrestre fica bastante agradecido e convida Murphy para lhe fazer uma visita e tomar um café intergaláctico. O progresso do jogo é salvo e o personagem inicia a viagem para o planeta Vulcanium. Quando Murphy chega ao planeta, acontece a cena final do jogo, em que os dois personagens tomam café intergaláctico e viajam na nave de Uba-Spock em direção à Terra. Ao aterrizar, Murphy é promovido para piloto profissional estagiário.

III. Implementação

O jogo Murphy foi desenvolvido com o uso do motor de jogo Unity. Entretanto, outras ferramentas foram utilizadas para criar os componentes de jogo. Os elementos gráficos foram inicialmente criados em papel e posteriormente redesenhados de forma digital, utilizando a ferramenta GIMP. Após a criação dos objetos de cena, utilizou-se a ferramenta Unity para compor os cenários e programar os comportamentos esperados para cada objeto. Por fim, o jogo incorporou arquivos de áudio, elaborados através do software livre Audacity[2], e incluídos ao jogo por meio do Unity.

O GIMP foi escolhido para criar os elementos de cena por ser um software livre e por exportar as imagens em extensões compatíveis com o Unity. Outra vantagem da ferramenta é a possibilidade de criar animações utilizando sequências de imagens. A Fig. 5, por exemplo, mostra o processo de destruição de um alvo ao receber um tiro.

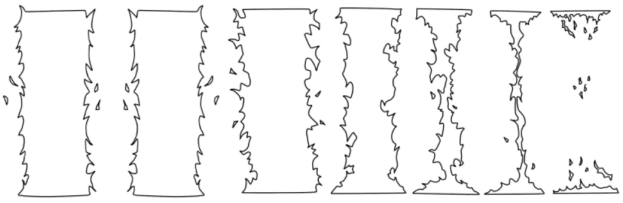

Fig. 5. Sequência da animação de destruição do alvo.

Por sua vez, os arquivos sonoros foram produzidos com o uso da ferramenta Audacity. Através dessa ferramenta criou-se todos os áudios incorporados ao jogo, ou seja, tanto as trilhas sonoras (menu, fases primárias e secundárias) como os efeitos de sons (tiro, acerto ou erro do alvo, interação entre os personagens, entre outros). O próximo passo foi unir os elementos de jogo, com o objetivo de desenvolver as fases e programar o comportamento dos componentes de cena. Para isso, criou-se um novo projeto no Unity, chamado Murphy. Esse projeto recebe tanto os componentes de jogo que já foram

[2]Disponível em: https://www.audacityteam.org

criados em outras ferramentas, como também armazena os arquivos criados pelo Unity. Logo após, criam-se novas pastas, uma para cada tipo de componente do jogo (animações, cenas, *scripts*, texturas, etc), a fim de manter a organização e facilitar o acesso. Todos os arquivos que compõem o jogo podem ser visualizados e manipulados dentro da aba `Project`, como mostra a Fig. 6.

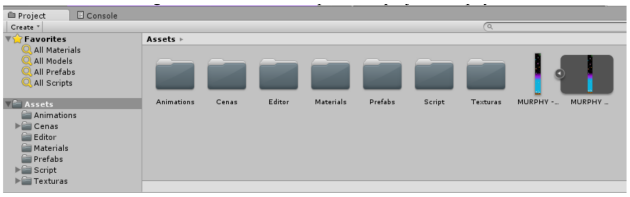

Fig. 6. Pastas com arquivos do projeto Murphy.

As cenas do jogo são construídas usando aba `Scene` e visualizadas em tempo real na aba `Game`. O que faz o objeto de cena aparecer na aba `Game` é a presença de uma câmera a qual define o espaço de tela que o jogador visualizará quando o jogo for renderizado. Tudo o que estiver dentro da área de alcance da câmera é o que será apresentado na tela.

Como ilustrado na Fig. 7, a área apresentada do cenário de uma fase do Murphy é maior do que a área mostrada na aba `Game`. Isto acontece porque a câmera está configurada para captar apenas um determinado espaço em torno do personagem. O cenário e outros elementos de cena aparecem a medida que o personagem sobe no *level*. A configuração da câmera é um dos primeiros elementos a serem ajustados, pois ajuda a visualizar como cada componente se apresentará ao jogador. O botão `Play` é responsável por iniciar a simulação do jogo e permite testar se as configurações utilizadas nos elementos estão funcionando corretamente sem requisitar salvar e compilar o jogo. Os elementos que estão inseridos na cena são catalogados na aba `Hierarchy` e podem ser configurados na aba `Inspector`.

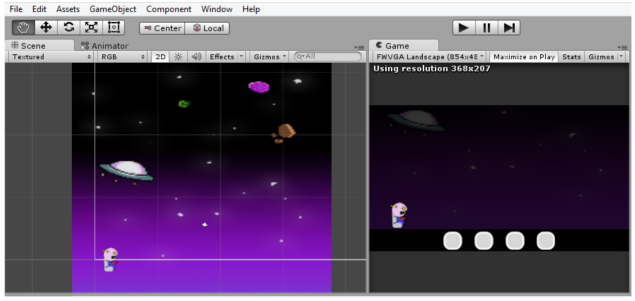

Fig. 7. Criação de um nível do modo viagem.

A aba `Inspector`, na Fig. 8, mostra as configurações do `Player`, elemento correspondente ao personagem inserido na `Scene` e listado na `Hierarchy`. Os elementos de cena terão a propriedade `Transform`, a qual define a posição, rotação e escala dos objetos. Se um objeto estiver muito pequeno na tela, basta configurar a altura no eixo x, a largura no eixo y e a profundidade no eixo z. Há uma relação de filiação entre o `Player`, o `NewBridgeSpawner` e um `GameObject` chamado `Bridge&Player`. Um `GameObject` é um elemento "invisível" que pode ser criado para representar objetos. O `Player` representa o personagem, o `NewBridgeSpawner` os trilhos e o `Bridge&Player` funciona como um recipiente desses componentes. Por exemplo, ao modificar o tamanho do personagem e do trilho numa proporção basta modificar o componente `scale` do objeto pai, ou seja, não é necessário mudar o `Player` e o `NewBridgeSpawner` individualmente, muda-se apenas o `Bridge&Player`.

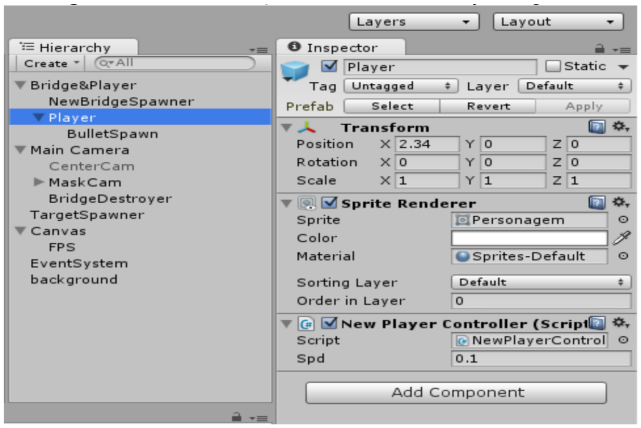

Fig. 8. Demonstração das abas Hierarchy e Inspector.

Outro componente do personagem mostrado na Fig. 8 é o `Sprite Renderer`. Ele é usado para processar a imagem que caracteriza o personagem. Se no decorrer do projeto optar-se por trocar a imagem do personagem sem alterar suas configurações, bastaria localizar a nova imagem na aba `Project`, arrastá-la e soltá-la no item `Sprite` do componente `Sprite Renderer`. Ou modificá-la em tempo de jogo através de um *script*.

Os *scripts* são criados no Unity, no menu `Create` da aba `Project`, e escritos na ferramenta de desenvolvimento `MonoDevelop`, que é instalada junto ao motor de jogo. A linguagem de programação adotada é C#. Cada *script*, quado criado, possui no mínimo duas funções: `Start` e `Update`. A função `Start` executa uma única vez sempre que o objeto é instanciado e a função `Update` executa enquanto o objeto existir. A principal ação do Murphy é disparar um tiro em linha horizontal sempre que houver um toque na tela, quando estiver numa fase de viagem.

Para isso, criou-se um `GameObject`, chamado `BulletSpawn`, posicionado com os mesmos valores que a arma do personagem, para simular o efeito de tiro. Esse objeto é filiado ao objeto `Player` para garantir que, caso haja mudanças na posição ou tamanho do personagem, por exemplo, o tiro sofra a mudança. Criou-se um *script*, adicionado como um componente de `BulletSpawn`, responsável pela ação do disparo. Sempre que houver um toque na tela, aparece em cena um novo objeto, um

pequeno feixe de luz que representa o tiro. A cor do tiro é determinada pela cor do trilho que está sustentando o personagem no momento do toque. O tiro inicia na posição que o `BulletSpawn` está e termina após um certo tempo, definido também no *script*. Logo, tem-se o efeito de que o personagem está atirando a cada vez que o jogador toca na tela. A Fig. 9 retrata o *script* que representa essa ação.

```
public class NewShot : MonoBehaviour{
  public static NewShot instance {get;protected set;}
  public Sprite BulletSpr;

  float cd = 1.75f;
  bool haidara {get {return Touched(1f) || Clicked(1f);}}
  NewGameControl ngc{get {return NewGameControl.instance;}}
}
```

Fig. 9. Criação da classe de tiro e definição de suas variáveis.

A classe `NewShot` é responsável pela criação e destruição do objeto referente ao tiro. A variável estática `instance` é uma instância dessa classe e é utilizada para facilitar o acesso ao objeto por outras classes. A variável `BulletSpr`, do tipo `Sprite`, armazena a imagem correspondente ao tiro. O tiro é criado via *script*, sendo executado e destruído somente em tempo de jogo. Ele precisa ser criado quando o jogador chamar a ação através do toque na tela. A variável `cd` recebe o valor $1.75f$ o qual determina a quantidade de segundos que o tiro ficará aparecendo na tela. Após esse tempo, o objeto está programado para ser destruído. A variável `haidara` do tipo booleano armazena o retorno dos métodos `Touched` e `Clicked` que retornam um valor quando o jogador realiza um toque na tela ou o clique do mouse. A variável `ngc` acessa e armazena o retorno da variável `instance` da classe `NewGameControl`, que controla quando o jogo começa e termina, ou seja, a variável `ngc` é utilizada para verificar se o jogo está iniciado ou não.

Os métodos `Clicked` e `Touched`, descritos na Fig. 10, retornam um valor verdadeiro ou falso. O primeiro é utilizado para controlar o clique efetuado pelo mouse em um objeto de cena. O segundo é utilizado para controlar o clique efetuado através do toque, em uma tela *touchscreen*. Assim, garante-se a possibilidade de experimentar o jogo tanto através do computador, como através de dispositivos sensíveis ao toque.

```
public bool Clicked (float per){
  float scr_ad = Mathf.Abs(ExtraMath.ScrnVector(false).x - ExtraMath.ScrnVector ().x) * per;
  if (ExtraMath.MousePos().x < ExtraMath.ScrnVector(false).x + scr_ad && Input.GetMouseButtonDown(0)){
    return true;
  }
  return false;
}

public bool Touched (float per){
  if (Input.touchCount > 0){
    Touch t = Input.GetTouch (0);
    if (t.position.x < (Screen.width * per) && t.phase == TouchPhase.Began){
        return true;
    }
  }
  return false;
}
```

Fig. 10. Métodos que controlam a interação do jogador com o objeto.

Conforme demonstrado na Figura 11, o método `Update` é executado a cada *frame*, durante todo tempo de jogo. A `ngc` é uma variável que armazena se o jogo está iniciado ou finalizado através dos métodos `GameEnd` e `GameStart`. Caso exista um tiro em cena, será invocado o `Destroy` e o tiro será eliminado. A variável `cd` receberá o valor de `Time.deltaTime`, que controla a quantidade de *frames* por segundo. Caso o tempo de `cd` seja maior que $1.75f$ e `haidara` seja verdadeiro, ou seja, se houve uma ação de clique ou toque do jogador, um novo tiro será criado e a variável `cd` receberá valor 0. Como o método `Update` é executado a cada *frame* e a variável `Time.deltaTime` controla os *frames* por segundo, um novo tiro só será criada quando o jogador executar um toque na tela, depois do tempo de $1.75f$.

```
void OnEnable(){
  instance = this;
}

void Update(){
  if (ngc.GameEnd () || !ngc.GameStart){
    if (GameObject.Find("Bullet") != null){
        Destroy(GameObject.Find("Bullet").gameObject);
    }
    return;
  }

  cd += Time.deltaTime;
  if (cd > 1.75f && haidara){
    CreateBullet();
    cd = 0;
  }
}
```

Fig. 11. Método Update.

O método `CreateBullet`, definido na Fig. 12, é o responsável por criar, dar forma e ação ao tiro, ou seja, o método cria e define o comportamento do `GameObject` chamado `blt`. Ao criar um `blt`, via *script*, um elemento `GameObject` é criado na cena do jogo quando o método `CreateBullet` for invocado. A variável `blt_spr` do tipo `SpriteRenderer` armazenará a imagem que foi fornecida ao objeto `blt`. Após definir a imagem que irá aparecer na tela, representada através do objeto `blt`, é preciso definir a cor do objeto. De acordo com o projeto, a cor do tiro é determinada pela cor do trilho que sustenta o personagem. Portanto, a cor será recebida através do acesso à classe que cria e define o trilho. Ou seja, a variável `blt_spr` receberá o valor (cor) da variável `curClr` do objeto que foi criado na classe `NewBridgeSpawner`.

Após criar e definir a cor do tiro, o método `CreateBullet` insere dois componentes ao objeto: `Rigidbody2D` e `BoxCollider2D`. Através desses componentes, possibilita-se inserir propriedades físicas aos elementos. O `Rigidbody2D` adiciona ao elemento uma propriedade de massa, fazendo com o que os objetos sofram a influência de forças físicas. Entretanto, é preciso definir uma área de colisão para o objeto, ou seja, adicionar o `BoxCollider`. O `BoxCollider2D` é responsável por

```
void CreateBullet (){
  GameObject blt = new GameObject("Bullet");
  SpriteRenderer blt_spr = blt.AddComponent<SpriteRenderer>();
  blt_spr.sprite = BulletSpr;
  blt_spr.color = NewBridgeSpawner.instance.curClr;
  blt.AddComponent<Rigidbody2D>();
  blt.AddComponent<BoxCollider2D>();
  blt.collider2D.isTrigger = true;
  blt.rigidbody2D.isKinematic = true;
  blt.transform.position = transform.position;
  blt.AddComponent<NewBullet>();
}
```

Fig. 12. Método CreateBullet.

detectar colisões entre elementos que possuem corpo. Através da inserção desses componentes, pode-se destruir o objeto correspondente ao alvo no momento em que o tiro colide. A função `isTrigger` detecta a colisão, ou seja, garante que os elementos não se espalhem pela cena no momento do choque. A função `isKinematic` garante que o objeto pare de reagir às forças aplicadas, ou seja, o tiro irá seguir em linha reta. Se houver a ausência desse componente o tiro, ao ser "disparado" iria sofrer a força da gravidade e iria "cair".

IV. Lições aprendidas

O processo de desenvolvimento do jogo eletrônico Murphy foi dividido em três etapas: i) conceituação e a idealização das características, regras e objetivos do jogo; ii) documentação das ideias no *Game Design Document* (GDD) e iii) implementação. Criar um jogo eletrônico é uma atividade complexa e multidisciplinar, que apresenta diferentes desafios no processo de construção.

Um desafio inicial foi a aquisição de conhecimento sobre a plataforma Unity. Por se tratar do primeiro contato com a plataforma, demandou-se tempo para habituação com o ambiente de desenvolvimento. Para adquirir esse conhecimento, foi imprescindível o estudo da linguagem C#, além da análise da documentação disponível no site do Unity e tutoriais disponibilizados em sites especializados.

A implementação foi orientada com base no GDD. Porém, apenas a adoção deste documento demonstrou-se insuficiente e, consequentemente, percebeu-se uma carência de um processo melhor alicerçado por boas práticas de engenharia de software como, por exemplo, integração contínua e desenvolvimento orientado a testes. Em particular, a adoção de diagramas e modelos *Unified Modeling Language* (UML) poderia mitigar algumas falhas de compreensão e comunicação entre os *stakeholders* do projeto. Por sua vez, a plataforma Unity demonstrou-se compatível com outras ferramentas utilizadas no processo de construção. Tal aspecto facilitou todo o processo, pois a equipe pôde elaborar diversos objetos de modelagem em ferramentas especializadas, e enviá-los à *engine*. Assim, constatou-se que a Unity se mostrou eficaz e suficiente para o processo de integração dos componentes do jogo.

V. Considerações Finais

Este artigo teve como objetivo relatar o processo de construção do jogo Murphy, desenvolvido para a plataforma Android, com o uso do motor de jogo Unity. Diante desse contexto, foram detalhadas questões pertinentes ao GDD, como *storytelling*, personagens, jogabilidade, elementos de cena e *level design*. Além de prover um detalhamento sobre o processo de *design* e ideação de um jogo casual, este trabalho agrega uma contextualização técnica sobre a adoção de diferentes tecnologias, com ênfase no Unity. Por fim, discutiu-se uma série de lições aprendidas com o projeto os quais podem contribuir na discussão junto a comunidade sobre o processo de desenvolvimento de jogos.

Percebe-se que a complexidade de desenvolvimento de um jogo é proporcional ao seu tipo, gênero e mecânica. Em particular, para o desenvolvimento do Murphy não foram necessários recursos ou conhecimentos matemáticos avançados, a Unity e demais ferramentas auxiliares *open source* contemplaram tais requisitos. Em termos de trabalhos futuros, pretende-se exportar o jogo Murphy para outras plataformas (iOS e PC) e continuar aprimorando a experiência do usuário e o *design* do jogo.

Agradecimentos

Os autores deste artigo agradecem a Robson Lima, Ronney Santos, Thaís Firmino, Daniel Souza, Vanderlane Lima e todos os demais colegas pela fundamental colaboração na jornada de desenvolvimento do Murphy.

Referências

[1] Game Brasil. Pesquisa Game Brasil. *Disponível em: https://www.pesquisagamebrasil.com.br*, 2020.

[2] Unity. Unity - Our Company. *Disponível em: https://unity.com/our-company*, acesso em julho de 2020.

[3] D. V. Macedo and M. A. F. Rodrigues. Experiências com desenvolvimento ágil de um jogo casual para plataformas móveis usando o motor gráfico unity. *Proceedings of the XI SBGames*, 2012.

[4] E. M. Chaves and R. L. F. de Ávila. Blind Runner: game design de um jogo corrida infinita acessível à cultura com deficiência visual. *Proceedings of the XVI SBGames*, 2017.

[5] F. Tołkaczewski. From symbolism to realism. physical and imaginary video game spaces in historical aspects. *Homo Ludens*, 2019.

[6] J. Á Vallejo-Pinto, J. Torrente, M. Ortega-Moral, and B. Fernández-Manjón. Applying sonification to improve accessibility of point-and-click computer games for people with limited vision. In *Proceedings of the 25th Conference on Human Computer Interaction*, 2011.