UNIVERSIDADE FEDERAL DO MARANHÃO

Curso de Ciência da Computação

Guilherme de Oliveira Lima

Framework para Construção de Jogos de Cartas Colecionáveis com RA

São Luís - MA

Fevereiro - 2018

Guilherme de Oliveira Lima

Framework para Construção de Jogos de Cartas Colecionáveis com RA

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Curso de Ciência da Computação

Universidade Federal do Maranhão - UFMA

Orientador: Prof. Dr. Geraldo Braz Junior

São Luís - MA

Fevereiro - 2018

Ficha gerada por meio do SIGAA/Biblioteca com dados fornecidos pelo(a) autor(a).
Núcleo Integrado de Bibliotecas/UFMA

Lima, Guilherme.
Framework para Construção de Jogos de Cartas Colecionáveis com RA / Guilherme Lima. - 2018.
42 f.

Orientador(a): Geraldo Braz.
Monografia (Graduação) - Curso de Ciência da Computação, Universidade Federal do Maranhão, São Luís, 2018.

1. Jogos. 2. Jogos 3D. 3. Jogos de Cartas Colecionáveis. 4. Jogos em RA. 5. Realidade Aumentada. I. Braz, Geraldo. II. Título.

Guilherme de Oliveira Lima

Framework para Construção de Jogos de Cartas Colecionáveis com RA

Monografia apresentada ao curso de Ciência da Computação da Universidade Federal do Maranhão, como parte dos requisitos necessários para obtenção do grau de Bacharel em Ciência da Computação.

Trabalho APROVADO em São Luís - MA, 22 de Fevereiro de 2018:

Prof. Dr. Geraldo Braz Junior
Orientador

Prof. Dr. João Dallyson Sousa de Almeida
Examinador

Prof. Dr. Anselmo Cardoso de Paiva
Examinador

São Luís - MA
Fevereiro - 2018

Resumo

O desenvolvimento de aplicações de realidade aumentada tem crescido nos últimos anos, em parte pela diminuição dos custos das tecnologias utilizadas nessa área, com isso os desenvolvedores de jogos ganharam uma nova forma de criar jogos que possibilitem uma imersão maior por parte dos jogadores e uma nova forma de interação com os objetos virtuais, que agora ocupam o ambiente real. Com o lançamento do jogo Pokemon GO para smartphones, os jogos de realidade aumentada ganharam um foco ainda maior. Um ramo dos jogos que têm muito campo para exploração são os jogos de cartas colecionáveis, como Magic e Yu-Gi-Oh!, que embora já tiveram implementação em realidade aumentada para este estilo de jogo, nenhum deles ainda teve grande destaque com o público, fazendo com que este campo ainda seja bem explorado pelos desenvolvedores.

Contudo desenvolver jogos de realidade aumentada pode não ser uma tarefa fácil, embora existam game engines que já agilizam o processo de desenvolvimento, o processo de criação de sistemas de jogabilidade e criação de todos os componentes que representam o comportamento de jogadores e cartas requerem conhecimentos variados (animações, modelagem, programação, calibração de câmera, entre outros), além de serem demorados e estarem sujeitos a erros no desenvolvimento.

O presente trabalho tem por objetivo oferecer um framework para agilizar o desenvolvimento de jogos de cartas colecionáveis com realidade aumentada, permitindo que o desenvolvedor crie jogos rapidamente, minimizando as possibilidades de erros durante o processo de construção do seu jogo, além de não requer conhecimento de programação por parte do desenvolvedor.

O framework é intitulado de ADGAR, sigla para Automation of the Development of Games with Augmented Reality (que significa Automatização do Desenvolvimento de Jogos com Realidade Aumentada), desenvolvido para trabalhar na Unity e usa a biblioteca do Vuforia para tratar os componentes de realidade aumentada.

O ADGAR Framework oferece um sistema de jogabilidade pré-definido, assim o desenvolvedor de jogos não precisa se preocupar em como criar modelos de jogabilidades que executem eletrônicamente. Como está integrado com a Unity é possível utilizar os modelos disponíveis no Asset Store (Base de dados online da Unity que possui livre acesso, onde pode-se adquirir modelos 3D e outros recursos).

Palavras-chaves: Jogos, Jogos em RA, Realidade Aumentada, Jogos 3D, Jogos de Cartas Colecionáveis.

Abstract

The development of augmented reality applications has grown in recent years, in part by lowering the costs of the technologies used in this area, so game developers have gained a new way of creating games that allow for greater immersion by players and a new interaction with virtual objects, which now occupy the real environment. With the launch of the Pokemon GO game for smartphones, augmented reality games have gained even greater focus. One branch of the games that have a lot of field for exploration are collectible card games, such as Magic and Yu-Gi-Oh !, which although they have already had an augmented reality implementation for this style of play, none of them still had a great deal of public attention , making this field still well explored by the developers.

However developing augmented reality games may not be an easy task, although there are already game engines that speed up the development process, the process of creating gameplay systems and creating all the components that represent the behavior of players and cards require a variety of knowledge (animations, modeling, programming, camera calibration, among others), as well as being time consuming and subject to development errors.

The objective of this work is to offer a framework to accelerate the development of collectible card games with augmented reality, allowing the developer to create games quickly, minimizing the possibility of errors during the process of construction of his game, and does not require knowledge of programming by the developer.

The framework is titled ADGAR, an acronym for Automation of the Development of Games with Augmented Reality, which is designed to work with Unity and uses the Vuforia library to handle augmented reality components.

The ADGAR Framework offers a pre-defined gameplay system, so the game developer does not have to worry about how to create game models that run electronically. Since it is integrated with Unity, it is possible to use the models available in the Asset Store (free online database where you can purchase 3D models and other resources).

Keywords: Games, Games in AR, Augmented Reality, 3D Games, Trading Card Game.

Lista de ilustrações

Sumário

1 Introdução

O mercado de jogos está constantemente nas mídias (jornalísticas, econômicas e sociais) e cresce significativamente, no Brasil o número de empresas que desenvolvem jogos cresceu cerca de 600%, chegando a aproximadamente 300 empresas de games, e o faturamento desse mercado no país era estimado em US$ 1,28 bilhão, em 2014, subindo para US$ 1,6 bilhão em 2016 (um aumento de 25% no faturamento) (SILVEIRA, 2017). Uma área em particular ganhou maior notoriedade recentemente, estes são os jogos em Realidade Aumentada (RA), embora as pesquisas em RA já existirem há algum tempo, inclusive aplicado a jogos (MALHEIROS; REIS; CARVALHO, 2012).

Em um ramo desse mercado encontram-se os jogos de cartas colecionáveis, como Yu-Gi-oh! e Magic. Um dos problemas para este tipo de jogo é que o combate em si fica a cargo da imaginação dos jogadores (MALHEIROS; REIS; CARVALHO, 2012), diminuindo a emoção do jogo vivenciada pelos jogadores e criando uma certa resistência do público mais novo por este tipo de jogo.

Com a aplicação de RA para jogos de cartas colecionáveis obteve-se uma melhor experiência por parte dos jogadores, como mostrado em (MALHEIROS; REIS; CARVALHO, 2012), e diminuição na resistência por parte do público mais novo, que pode assistir às batalhas através de um recurso visual.

Existem iniciativas na área de jogos de cartas colecionáveis, como o Invizimals que foi desenvolvido inicialmente para PSP, necessitava, além do próprio PSP, uma câmera portátil (PANINI, 2017). Permite a visualização das cartas em RA no console ou aparelho no qual foi instalado (PANINI, 2017). As cartas são detectadas através de código que combina quadrados no verso da carta (Figura 1).

Figura 1 – Invizimals cartas e no celular

Source: (PANINI, 2017)

Já o Eye of Judgment, produzido exclusivamente para PS3 onde os jogadores utilizam cartas reais posicionando-as em lugares específicos (GALAM; DIAS, 2017), a RA

adiciona a arena e os modelos 3D. O sistema de jogo, diferente dos modelos tradicionais, consiste em conquistar 5 dos 9 campos disponíveis na partida (SONY, 2007). Não atingiu um grande público pois além da necessidade de comprar as cartas era necessária a aquisição de uma câmera própria pro console, que aumentava consideravelmente o preço do jogo (GALAM; DIAS, 2017), e do tabuleiro do jogo (VINHA, 2016).

Figura 2 – Jogo Eye of Judgment

Source: (SONY, 2007)

Além dos jogos, existem as bibliotecas OsgART (OSGART NG, 2009), ARToolKit (ARTOOLKIT, 2004) e C.A.V.E.I.R.A. (MARON; MARSON, 2009) onde jogos de cartas podem ser desenvolvidos juntamente com a tecnologia de RA.

O desenvolvimento desses jogos requer conhecimento (nas áreas de modelagem 3D, animações, processamento de imagens, comunicação de hardwares, entre outros), mesmo que muitas funcionalidades possam ser reaproveitadas por serem semelhantes (Movimentação, chamada de animações, execução de ações, cálculos) bem como o padrão de comportamento dos jogadores (jogador da vez ataca, sequencia pré-estabelecida de ações que podem ser tomadas, decremento dos pontos de vidas, etc). Todavia, também existem suas particularidades, como nome da carta ou do jogador, qual jogador jogou determinada carta, valores de atributos são específicos de cada carta, entre outras. Para um conjunto de pequenas cartas essa tarefa é simples, mas já se torna algo cansativo, levando em consideração um conjunto maior pode ser bem demorado desenvolver o jogo, fora a alta probabilidade de acontecer erros no processo.

Este trabalho visa fornecer um framework de automatização do desenvolvimento de jogos de cartas colecionáveis com RA que diminua os riscos de erro durante a construção do jogo, necessitando do jogador pouco ou nenhum conhecimento de programação.

Especificamente, desejamos fornecer um sistema de jogabilidade pré-definido, evitando que o desenvolvedor demande muito tempo pensando em sistemas que sejam computáveis, neste sistema a opção de ataque e pontos de vida são obrigatórios, o de-

senvolvedor poderá optar no tipo de pontos de vida e se inclui: pontos de vida na carta, custo de colocar uma carta e esquiva da carta [1]; abstrair a parte de programação para o usuário, gerando automaticamente os arquivos necessários para o funcionamento do jogo, como controle de turno [2], inserção de monstros, mecânica de jogo, animação do modelo 3D, comportamento da carta, etc. Poupando o tempo de alteração de código em cada arquivo ao ser colocado uma nova carta na construção do jogo e agilizar o processo de desenvolvimento do jogo, através do sistema de automatização de sua criação e o uso de uma plataforma que ofereça recursos prontos sem que o desenvolvedor precise estar criando algo do zero, como modelos 3D, animações, entre outras possibilidades.

1.1 Estrutura do Trabalho

O **Capítulo 1: Introdução** apresenta a motivação do trabalho, os objetivos, trabalhos relacionados com as temáticas abordadas neste trabalho (Jogos de cartas colecionáveis com RA, frameworks para desenvolvimento de jogos, bibliotecas para RA e frameworks para desenvolvimento de cartas colecionáveis com RA).

O **Capítulo 2: Fundamentação Teórica** traz conceitos para apresentar e definir os assuntos e tecnologias utilizados neste trabalho. É apresentado a definição e evolução da RA, a história dos jogos de cartas colecionáveis, definições e alguns dos Padrões de Projeto conhecidos, conceito de frameworks e por fim uma apresentação da Unity e do Vuforia.

O **Capítulo 3: Modelagem do ADGAR Framework** mostra o desenvolvimento da ferramenta, como ela está dividia, as estratégias aplicadas no decorrer do desenvolvimento e suas funcionalidades.

O **Capítulo 4: Estudo de Caso - Apocalipse Mitológico** traz um jogo modelado pra validar o framework desenvolvido por este trabalho, faz uma apresentação ao jogo, como foi desenvolvido na ferramenta e as conclusões do trabalho.

O **Capítulo 5: Conclusões** apresenta as considerações deste trabalho e propostas de melhorias e evolução da ferramenta.

[1] quando uma carta é atacada ela se desvia do ataque

[2] Os jogadores se alterna pra jogar, quando um joga o outro fica esperando, como acontece em jogos de damas

2 Fundamentação Teórica

Este capítulo tem por finalidade apresentar a base teórica utilizada para o desenvolvimento deste trabalho e dar um entendimento ao leitor dos temas tratados. São abordados nas seções deste capítulo os conceitos de RA, jogos de cartas colecionáveis, Padrões de Projeto, framework, Unity, Vuforia.

2.1 Jogos de Cartas Colecionáveis

Os jogos de cartas colecionáveis são jogos de estratégia e sorte que consiste na formação de um deck (quantidade de cartas conforme o mínimo e máximo estão dentro das regras do jogo), onde as cartas usadas fica a critério do jogador, cujo objetivo é derrotar o(s) oponente(s), necessitando de dois ou mais jogadores que se enfrentam em turnos alternados (CAROLINE, 2017); além de possibilitar a coleção e trocas das cartas (GARFIELD et al., 2013).

Os jogos de cartas surgiram a muito tempo, sendo que o registro mais antigo é de 1294 na China (COPAG, 2018). Com o passar do tempo o jogo sofreu várias alterações e variações até chegar a forma dos *trading cards* que se tornaram muito populares nos Estados Unidos a partir do fim do século XIX, estas cartas eram feitas sobre diversos temas e traziam informações estatísticas sobre este tema (JACINTHO, 2013).

Os jogos de cartas colecionáveis com o modelo atual surgiu no ano de 1993, com o lançamento do jogo Magic, combinando os trading cards e jogos de cartas (daí a origem do nome em inglês: Trading Card Game - TCG). Desde então o estilo de jogo se popularizou e surgiram diversos outros jogos do ramo com suas próprias regras, dentre os mais conhecidos temos Yu-Gi-Oh! e Pokemon (JACINTHO, 2013).

Até hoje há novos jogos do gênero surgindo com temáticas diferentes ou mesmo tentando inovar na sistemática de jogos, jogos de cartas colecionáveis puramente virtuais (CAROLINE, 2017) ou outros meios de difundir e ampliar a experiência do jogo. E neste contexto de novas abordagens entra a RA como uma alternativa inovadora de interface e interação com o jogo (TORI et al., 2007), extraindo a batalhas do campo da imaginação para um ambiente visual (MALHEIROS; REIS; CARVALHO, 2012).

Dentre os jogos desenvolvidos com RA temos como exemplo o Invizimals, o Eye of Judgment (apresentados no Capítulo 1), o jogo Batalha dos Mitos, desenvolvido na Bahia, cuja temática do jogo aborda elemnentos dos mitos brasileiros (MELO, 2015) e o ARBattle prosposto por Malheiros, Reis e Carvalho (2012).

2.2 Realidade Aumentada (RA)

A RA faz com que o mundo virtual seja inserido no mundo físico, em tempo real, facilitando a interação das pessoas com objetos virtuais (KIRNER; TORI, 2006). A visualização destes objetos virtuais não é possível através de olho nu, é preciso o auxílio de tecnologias próprias pra isso, como por exemplo monitores e óculos de realidade aumentada (MALHEIROS; REIS; CARVALHO, 2012).

É interessante ressaltar que a realidade aumentada é uma ramificação da realidade misturada, que visa misturar cenas virtuais com cenas reais (seja num ambiente real ou virtual), de tal maneira que o usuário não conseguiria distinguir os elementos implantados nos ambientes, formando um ambiente mais realista (KIRNER; TORI, 2006).

Botega e Cruvinel (2009) mostra que diferente da realidade virtual (RV) e da RM, a RA usa componente de custo mais barato e acessível às pessoas, como câmeras, marcadores de papel e técnicas de visão computacional, para proporcionar níveis de imersão ao usuário.

Assim, conforme Kirner e Tori (2006), a realidade aumentada pode ser classificada em imersiva ou não imersiva, dependendo da maneira como o usuário enxerga os objetos virtuais. No primeiro caso (imersivo) o usuário olha diretamente para o mundo real e vê os objetos virtuais misturado com o mundo, os objetos virtuais são vistos com o auxilio de equipamentos visuais transparentes (que possuem um custo mais elevado) (BOTEGA; CRUVINEL, 2009); já no segundo caso (não imersivo) o usuário olha o mundo real com os objetos virtuais através de algum dispositivo, como monitores.

A RA passou a ganhar mais ênfase no mundo dos jogos eletrônicos em meados do ano de 2002 devido ao barateamento dos hardwares usados, antes disso quando se falava em RA para entretenimento era com foco em aplicações no cinema. Com a maior aplicação da RA nos jogos eletrônicos percebeu-se um leque mais amplo de alternativas do uso desta tecnologia, como integrar usar a RA para jogos que ainda não possuíam uma mídia eletrônica, tais como os jogos de cartas (embora houvesse jogos eletrônicos com essa temática os jogadores ainda preferiam os jogos em papel), surgindo uma nova forma de interação e imersão no jogo (TORI et al., 2007). Um exemplo disso é mostrado por Miyazaki, Rey e Marengoni (2012), que apresenta uma aplicação de RA para jogos de RPG de mesa, projetando os elementos do jogo numa mesa e os jogadores interagem diretamente com essas projeções.

Entretanto, mesmo com os avanços tecnológicos e diminuição dos custos de hardware e software a maioria dos jogos com RA ainda não conseguiram um grande público, principalmente nos jogos de cartas (GALAM; DIAS, 2017).

2.3 Padrões de Projeto

Os Padrões de Projeto são soluções de implementação que resolvem um determinado problema através de padrões de reúso de código. Um padrão surge da experiência de várias pessoas que passaram por um mesmo tipo de problema no desenvolvimento de alguma aplicação computacional e resolveram esse problema de forma semelhante(BRIZENO, 2016a).

Uma definição mais formal é apresentado por Brizeno (2016b, p.16): "Padrão de projeto é uma solução comum para um problema em um determinado contexto".

Brizeno (2016a) mostra ainda uma divisão dos padrões de projetos conforme a natureza do problema a ser resolvido e os padrões listados pela Gangue dos Quatro (Gang of Four [1]) dentro dessa classificação, são eles:

Padrões de Criação: Abstrai o processo de criação/instanciação de objetos. O objetivo é que o sistema evolua de forma independente. Nesta classificação temos os Padrões Factory Method, Abstract Factory, Builder, Prototype e Singleton (BRIZENO, 2011).

Padrões Estruturais: Se preocupam com as estruturas das classes e objetos. O objetivo é facilitar o design e o relacionamento das entidades. Os padrões estruturais mais conhecidos são o Adapter, Bridge, Composite, Decorator, Facade, Flyweignt e Proxy (BRIZENO, 2011).

Padrões Comportamentais: Trata dos comportamentos dos objetos. O objetivo é facilitar a comunicação entre os objetos. Os padrões mais conhecidos dessa classe são : Interpreter, Template Method, Chain of Responsibility, Command, Iterator, Mediator, Memento, Observer, State, Strategy e Visitor (BRIZENO, 2011)

Foram utilizados nesse projeto os padrões *Prototype* e *Singleton* para melhoria da estrutura do código e manter a integridade das informações usadas pelas classes dos objetos.

A ideia do padrão *Prototype*, mostrado por Brizeno (2011), é definir um objeto como padrão e ao criar um novo objeto este será uma cópia do objeto padrão. No decorrer do desenvolvimento do framework observou-se comportamentos semelhantes entre as cartas e os demais elementos do jogo. Assim foi criado para cada script um elemento que serve de base para a construção deles, no caso de scripts gerais, como será mostrado no Capítulo 3.1.5, esse modelo base é copiado sem sofrer alterações. Já no caso da criação de scripts específicos as informações inseridas nas variáveis são diferentes para cada carta, não copiando o modelo totalmente igual, mas ainda assim seguindo a ideia do padrão *Prototype*.

[1] Documentaram em seu livro *Design Parterns: elements of reusable object-oriented software* 23 padrões de projeto

Já no caso do padrão *Singleton* a proposta é definir um único ponto de acesso para construção de objetos, mantendo a integridade das informações deste objeto (BRIZENO, 2011). Este padrão também é usado no *Módulo de Geração de Scripts* pois mais de uma classe precisa fazer chamada a este módulo, então é utilizado um método público em cada tipo de geração de scripts (gerais e específicos), de forma a garantir que todas a informações necessárias sejam fornecidas e os scripts sejam criados adequadamente.

Os Padrões de Projetos facilitam a reusabilidade do software, o seu entendimento, alterações e remoções de funções sem afetar o funcionamento do projeto todo. A principal desvantagem desses métodos é a complexidade da estrutura do código pode tomar, por isso é sempre bom verificar se realmente há a necessidade de aplicar determinado padrão, nem sempre a aplicação deles traz benefícios significativos para a aplicação. Uma boa abordagem é não escrever um código aplicando um Padrão, refatorar o código utilizando um Padrão que se aplique e melhore o código(BRIZENO, 2016a).

2.4 Framework

É um sistema de software, ou parte de um, que pode ser reusado para desenvolvimento de uma aplicação específica com o proposito de reduzir os custos de produção e elevando sua qualidade, abstraindo processos de baixo nível e provendo uma interface de alto nível ao usuário, permitindo que este dê maior foco a outros detalhes da aplicação (MADEIRA, 2001), no caso específico deste trabalho o usuário poderia focar em criar uma relação em que os pontos de vida do jogador chegue a zero antes que as cartas dele acabem, estipulando limite de cartas em um deck [2] e valor dos pontos de vida.

Os benefícios proporcionados pelos frameworks são: *modularidade*, ocultando detalhes de baixo nível do usuário através do uso de interfaces estáveis; *reusabilidade*, através da determinação de comportamentos genéricos que são reaplicados na criação de novos projetos; *extensibilidade*; e *controle* (MADEIRA, 2001).

O fato de utilizar padrões genéricos para permitir a reusabilidade faz com que frameworks sejam grandes consumidores dos Padrões de Projetos, incorporando por vezes mais de um padrão (MADEIRA, 2001).

Estão separados em três classes, conforme mostrado em Madeira (2001):

Frameworks de Infra-estrutura: Utilizado no desenvolvimento de infra-estrutura de sistemas portáteis e eficientes, tais como sistemas operacionais. Abstraem mecanismos como threading, dispositivos de entrada/saída e gerenciamento de memória.

Frameworks de integração (middleware): Utilizado para integrar aplicações distribuídas e tem por finalidade melhorar a habilidade dos desenvolvedores para a sua

[2] conjunto de cartas utilizadas em uma partida de cartas colecionáveis

infra-estrutura de software.

Frameworks de aplicação: Utilizados nos mais diversos domínios de aplicações, em geral aplicações comerciais onde o objetivo é fornecer um produto ou serviço para um usuário final.

O framework desenvolvido neste trabalho se encaixa na classe de frameworks de aplicação, pois objetiva fornecer um produto que agiliza o processo de desenvolvimento de jogos para um usuário (o desenvolvedor de jogos).

2.5 Unity

A Unity é uma ferramenta de desenvolvimento de jogos em 2D, 3D, VR e AR, muito popular entre os desenvolvedores pela facilidade de uso e a variedade de linguagens que se pode usar para programar, com licença gratuita para desenvolvimento de jogos não comerciais (UNITY, 2017).

Disponibiliza em sua Asset Store modelos e animações 3D gratuitas e aceita importação de modelos externos criados em diversas plataformas, como Blender e 3DS Max.

Fornece documentação on-line e off-line, tutoriais com vídeo, facilidade de portabilidade de código, podendo ser facilmente convertido para plataformas além do PC, como Android, IOS, Consoles, Samsung TV, entre outras (GALAM; DIAS, 2017).

Os jogos são construídos em cenas que nada mais são telas que compõem um jogo, por exemplo quando um personagem passa de fase em um jogo ele é encaminhado para uma nova cena. Dentro das cenas são inseridos os elementos que estarão presentes no jogo. Esses elementos são os *GameObjects*, um conjunto de componentes que vão caracterizar o jogo (elementos de mapa, personagens, NPCs, janelas de interface com o jogador). O *Asset Database* é a pasta onde estão todos os elementos que são possíveis de se inserir no jogo, temos os *Prefabs* que são elementos que já foram configurados previamente e o mesmo *prefab* pode ser usado várias vezes jogo (um *prefab* que representa um tijolo pode ser usado várias vezes para formar uma parede).

2.6 Vuforia

Vuforia é uma biblioteca de realidade aumentada com fácil integração à Unity Engine que facilita a captura de objetos do mundo real e o posicionamento do objetos virtuais sobre estes, fornecendo a possibilidade de criação da base de dados em seu site, para uma melhor interação com seus recursos.

O Vuforia tem licença gratuita para desenvolvimento não comercial, com licenças

disponíveis, também, para Android, iOS e UWP (Plataforma Universal do Windows), além da versão Unity (VUFORIA, 2017).

As bibliotecas para RA em geral usam por base marcadores com padrões quadrados, como os mostrados no capítulo anterior (Figura 1). O Vuforia utiliza marcadores naturais (GALAM; DIAS, 2017), reconhecendo as próprias imagens ao invés de combinações de quadrados. Possui um fórum de desenvolvedores na Unity.

O Vuforia apresenta uma ampla variedade de formas de se trabalhar com a RA (VUFORIA, 2017):

Model Targets: usados para reconhecer objetos por sua forma usando modelos 3D existentes;

Ground Plane: para colocar objetos em superfícies horizontais em seu ambiente;

Image Targets: para aplicações que usam imagens planas, como cartas.

VuMarks: para identificar e aumentar objetos específicos como parte de uma série, como brinquedos e produtos de consumo;

Multi-Targets: são coleções de *Image Targets* em um arranjo definido, utilizados para embalagens de produtos e outdoors.

Cylinder Targets: permite que use qualquer imagem cilíndrica em aplicativos RA;

User Defined Targets: permite que use imagens de câmera, capturadas pelos usuários como alvos de imagem;

Object Recognition: permite que os *Object Targets* sejam criados pela varredura de objetos físicos. Permite criar aplicativos que reconheçam e rastreiem objetos rígidos intricados.

O Vuforia ainda fornece suporte a RA+RM com *head tracking* e uma API de Realidade Mista.

Conforme mostra Vuforia (2017) a plataforma possui três componentes principais:

Vuforia Engine mecanismo de jogo de plataforma cruzada para criar aplicações, esta biblioteca está do lado do cliente e tem suporte ao Android Studio, Xcode, Visual Studio e Unity.

Tools ferramentas para criar alvos, gerenciar banco de dados de destino e garantir licenças de aplicações. Para óculos digitais pode-se usar a ferramenta *Calibration Assistant* permitindo aos usuários finais criar perfis personalizados que se adaptem a sua geometria facial exclusiva e o *Vuforia Engine* usa esse perfil para renderizar o conteúdo na posição correta.

Cloud Recognition Service uma ferramenta quando a aplicação precisa reconhecer um grande conjunto de imagens ou com atualização contante da base de dados.

Oferece muito material de suporte aos desenvolvedores no seu Developer Portal, com bastante conteúdo, tutoriais e notícias sobre as novas versões que são lançadas (VUFORIA, 2017).

O Vuforia é utilizado no framwork para captura do mundo real, rastreamento das imagens das cartas utilizadas (com o uso do ImageTraget) e posicionamento do modelos 3D no mundo real.

3 Modelagem do ADGAR Framework

O framework proposto neste trabaho foi intitulado de ADGAR, sigla para *Automation of the Development of Games with Augmented Reality*, que significa Automatização do Desenvolvimento de Jogos com Realidade Aumentada.

O ADGAR framework utiliza a biblioteca do Vuforia para tratar os componentes de RA. O Vuforia oferece aos seus usuários suporte a criação de uma base de dados, onde são inseridos as imagens que se deseja utilizar no jogo. O mesmo avalia a imagem submetida e indica, através de uma avaliação por estrelas, se é uma boa imagem para se usar ou não (a partir de 3 estrelas é possível que a câmera rastreie a imagem, abaixo disso a imagem é ruim e precisa ser melhorada), após as imagens submetidas é possível exportar a base para o tipo de aplicação desejada (Unity, Android, entre outros), para mais informações vide Capítulo 2.

O framework foi desenvolvido para a Unity3D, versão 5.6, devido sua fácil integração com o Vuforia, fácil portabilidade da aplicação e uma *Asset Store* ampla que permite ao desenvolvedor criar novos jogos a um baixo custo. A linguagem de programação utilizada foi o C# e o Vuforia 6.2. Antes do framework ser utilizado é preciso instalar o Vuforia no Unity.

O framework foi dividido em sete módulos, como mostra a Figura 3, afim de facilitar o desenvolvimento e alterações do código para melhorar e expandir a ferramenta. Os módulos são: ***Módulo de Interface Principal***, ***Módulo de Interface Configuração***, ***Módulo Geração da Base***, ***Módulo de Obtenção dos Atributos de uma Carta***, ***Módulo Geração de Scripts***, ***Módulo Geração de Controle de Animações*** e ***Módulo Interface de Finalização do Jogo***.

Os elementos *ImageTarget* e *ARCamera* são *prefabs* que já vêm com a biblioteca do Vuforia, sendo necessário apenas instanciá-los na cena do jogo, o *ImageTarget* guarda as informações da imagem de uma carta, já o *ARCamera* captura o mundo real e compara as informações obtidas com as informações em cada *ImageTarget*, se encontrar características semelhantes o Vuforia exibe o modelo 3D anexado ao *ImagetTarget*.

Há ainda elementos que são criados pelo framework, eles são os arquivos de atributos, o arquivo de configuração e os scripts. Os arquivos de atributos são os arquivos responsáveis por guardar as informações referentes a carta (Ponto de Ataque, Modelo 3D a ser utilizado para representar a carta no jogo). O arquivo de configuração guarda as informações que servem de regras para o jogo e a construção dele seguirá como roteiro estas regras, esse arquivo é nomeado de **Config** e gerado no ***Módulo de Interface Configuração***. Os scripts são arquivos de código que compõem os *GameObjects* definindo características e

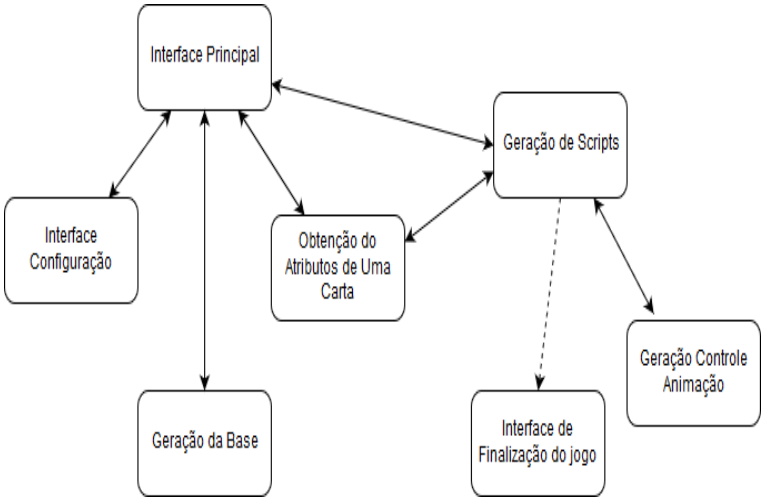

Figura 3 – Módulos do ADGAR Framework.

ajudando a determinar o comportamento dos *GameObjects* dentro do jogo.

Na primeira seção deste capítulo apresentamos funcionalidades gerais que são necessárias na descrição dos módulos. Na segunda parte, apresentamos o detalhamento dos módulos propostos no ADGAR.

3.1 Funcionalidades

O sistema de jogabilidade e arquivo de configuração e a configuração via arquivos das cartas são requisitos básicos para o funcionamento pretendido do framework. Ambos serão abordados a seguir.

3.1.1 Sistema de Jogabilidade e Arquivo de Configuração

O Sistema de Jogabilidade são todos os elementos que estarão presentes no jogo, funcionam como as regras do jogo. Os elementos possíveis são pré-determinados, mas o desenvolvedor pode escolher quais utilizar podendo criar sistemas de jogos variados para cada jogo.

A escolha do sistema de jogabilidade é definida no ***Módulo de Interface de Configuração***, na tela de interface basta o desenvolvedor selecionar as opções desejadas, conforme mostrado na seção anterior.

Os jogos de cartas colecionáveis desenvolvidos com esta ferramenta funcionam com a sistemática de turnos, enquanto um jogador está colocando cartas ou atacando o outro terá que aguardar até que o turno seja encerrado e passe para sua vez, esta opção não é editável pelo desenvolvedor.

Os elementos do Sistema de Jogabilidade são Tipo de Pontos de Vida do Jogador,

Ataque da Carta, Pontos de Vida da Carta, Custo de Jogar uma Carta, Velocidade da Carta. Abaixo estão explicados cada um desses elementos.

- **Tipos de Pontos de Vida do Jogador**

  Estáticos - modelo clássico dos jogos de cartas colecionáveis, onde os jogadores começam com uma valor de pontos de vida fixo, este valor é determinado pelo desenvolvedor ao escolher este tipo de ponto de vida. Neste modo os valores de ataque são comparados e o jogador com a carta com menor ponto de ataque sofre o dano, tendo os pontos de vida dele sendo reduzido, o valor reduzido é igual a diferença dos ataques. Caso o modo Pontos de Vida da Carta estiver habilitado, o dano é aplicado na carta e quando o valor dos pontos de vida dessa chegar a zero o valor base do ponto de vida da carta é deduzido dos pontos de vida do jogador.

  Dinâmico - as cartas são divididas em dois tipos, monstro e vida, as cartas de vida aumentam os pontos de vida do jogador e não podem ser usadas para atacar já que devem ter pontos de ataque igual a zero; já as cartas de monstros devem ser usadas para atacar. Neste modo quando uma carta ataca, o valor de ataque é subtraído diretamente dos pontos de vida da carta e quando chegar a zero o valor base dos pontos de vida da carta é diminuído nos pontos de vida do jogador.

- **Ataque** - esta opção é obrigatória, já está marcado e não pode ser desabilitado. Corresponde ao valor de ataque da carta, pode variar de 0 a 100. As cartas de tipo vida geralmente tem pontos de ataque zero enquanto as de monstro tem valores variados, mas isto depende do desenvolvedor.

- **Pontos de Vida da Carta** - Quando o *Tipo de Pontos de Vida do Jogador* for do tipo dinâmico, este atributo se torna obrigatório. Define um valor para representar os pontos de vida carta, o jogador só recebe dano quando uma carta for eliminada do campo de jogo. Os valores para esse atributo variam de 1 a 100. O dano aplicado ao jogador é igual aos pontos de vida base da carta eliminada.

- **Custo de Jogar uma Carta** - este valor é subtraído dos pontos de vida do jogador quando este joga uma carta, é um atributo opcional. Seus valores variam de 0 a 10. Como sugestão, o desenvolvedor pode utilizar este atributo para determinar a força da carta (quanto mais forte uma carta maior o custo de colocar ela).

- **Velocidade da Carta** - este é um valor que será atribuído à carta, no decorrer do jogo, quando uma carta ataca, um número aleatório é gerado para carta que ataca e um para carta que está sendo atacada, o valor do número aleatório é somado as respectivas velocidades das cartas e comparados, se o valor da carta atacante for maior o ataque acerta, caso contrário a carta que está sendo atacada desvia do ataque.

Quando o desenvolvedor escolhe os elementos do Sistema de Jogabilidade estes precisam ser salvos pois serão utilizados pelo framework e pelo jogo, então estes elementos são salvos no Arquivo de Configuração que define a estrutura como eles serão salvos. A Figura 4 mostra uma exemplo deste arquivo.

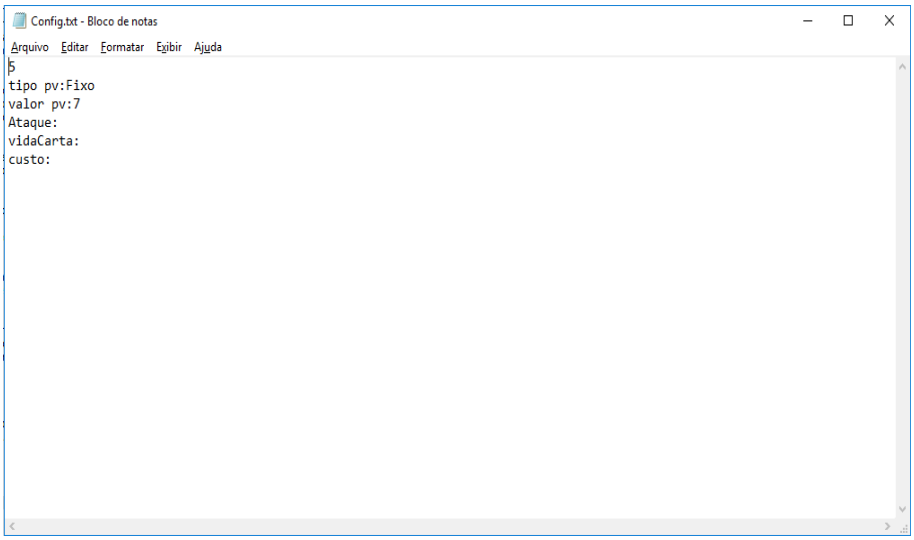

Figura 4 – Exemplo de um Arquivo de Configuração

O arquivo deve conter na primeira linha a quantidade de elementos do Sistema de Jogabilidade, as demais linhas são os elementos com seus valores. Cada elemento deve ser colocado em uma linha, primeiro com seu nome seguido de dois pontos e seu valor (no caso de não haver valor não é preenchido nada) não é utilizado espaços no arquivo.

O Arquivo de Atributos sempre é salvo na pasta criada pelo ADGAR Framework dentro do *Assets Database* e o desenvolvedor do jogo não precisa se preocupar com sua estrutura pois o sistema já salva o arquivo automaticamente.

3.1.2 Arquivos de Atributos de Uma Carta

São os arquivos que armazenam todas as informações a respeito de uma carta, estas informações dependem das configurações do Sistema de Jogabilidade, ou seja, estes elementos podem variar de jogo para jogo. O Arquivo de Atributo da Carta deve ter o mesmo nome da imagem e ficar salvo no mesmo local também.

Este arquivo pode ser gerado pelo framework ou pelo próprio usuário, mas deve seguir a seguinte estrutura:

- o nome do atributo vem primeiro seguido de dois pontos e do valor do atributo, sem espaços.

- o nome do atributo deve ser escrito com letra minuscula salvo quando o atributo tem dois nomes, neste caso o segundo nome começa com letra maiúscula, mas não há espaço entre os nomes.

- o modelo 3D da carta é referenciado pelo termo "Objeto"(deve ser escrito com a primeira letra maiúscula porque faz referência a um objeto no Assets Database) seguindo de dois pontos e o nome do modelo 3D. Este elemento sempre fica na última linha dos atributos.

Para observar melhor estas regras um exemplo do Arquivo de Atributos pode ser visto na Figura 5

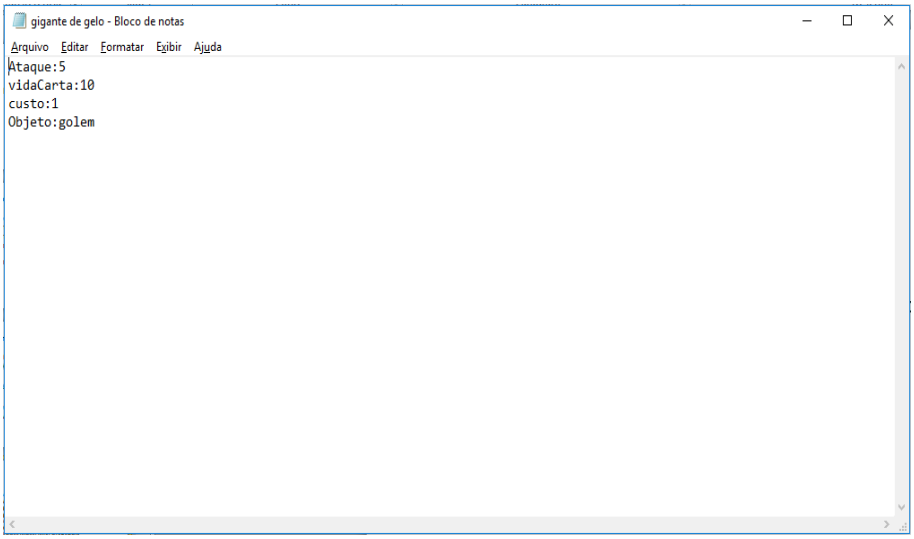

Figura 5 – Exemplo de um Arquivo de Atributos de uma Carta

3.2 Módulos

Nesta seção será apresentado com mais detalhes os módulos do ADGAR framework, suas descrições, as classes que compõe cada módulo e as estratégias que precisaram ser utilizadas e onde foram aplicadas para melhorar o projeto.

3.2.1 Módulo de Interface Principal

Este módulo fornece a principal interface para o usuário do ADGAR Framework, ele coleta as informações sobre quais imagens serão utilizadas no desenvolvimento do jogo, dos modelos 3D que serão utilizados para representar a carta e realiza chamada aos demais módulos, determinando quais serão executados e a ordem em que eles serão executados. É composto pela classe **TelaInicial** (Figura 6 mostra sua interface de usuário)

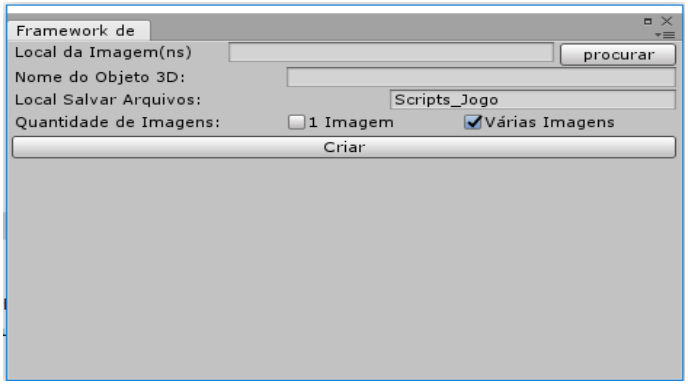

Figura 6 – Tela Principal do framework

O diagrama de atividades apresentado na Figura 7 traz uma descrição visual do funcionamento deste módulo, mostrando o conjunto de ações executadas e nos parágrafos seguintes é feita uma descrição de cada ação do diagrama.

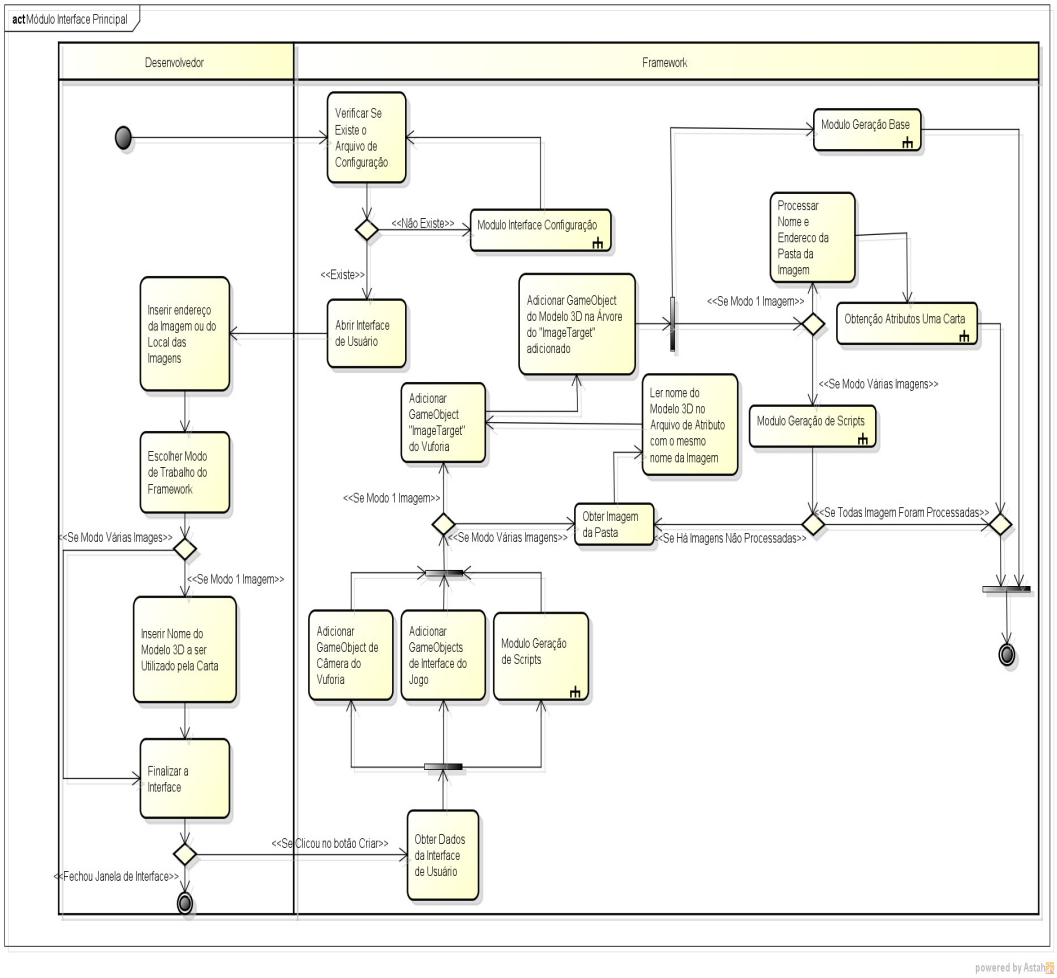

Figura 7 – Diagrama de Atividades do Módulo Interface Principal

Quando o usuário inicia o framework, o sistema verifica se há algum arquivo de configuração (nomeado de **Config**), no *Assets Database*. Se o arquivo não existe é

realizada uma chamada ao ***Módulo Interface Configuração***. Se o arquivo existe, o framework abre a janela de interface do usuário e aguarda até que a janela seja fechada ou o desenvolvedor do jogo clique no botão Criar.

O desenvolvedor Insere o endereço de uma pasta com várias imagens ou o enderenço de uma única imagem (pasta onde a imagem está salva + nome da imagem + formato da imagem), depois Seleciona o modo como o sistema irá trabalhar, se é no modo 1 Imagem ou no modo Várias Imagens. Se o desenvolvedor marcou o modo 1 Imagem ele deverá informar o nome do modelo 3D para essa imagem.

Se o desenvolvedor marcou o modo 1 Imagem ele deverá informar o nome do modelo 3D para essa imagem.

Se o desenvolvedor fechar a janela o framework encerra, se clicou no botão Criar o framework obtém os dados fornecidos pelo desenvolvedor na janela de interface e então adiciona a cena do jogo o *prefab ARCamera* do Vuforia e os *GameObjects* de Interface do jogo (são os GameObjects de: Interface do Jogo com o jogador e o *GameObject* que gerencia as informações dos jogadores), também é realizada uma chamada ao ***Módulo Geração de Scripts*** para criar os scripts gerais do jogo.

A partir daqui a sequência de atividades serão explicadas conforme o modo escolhido pelo desenvolvedor.

1ºCaso: *Se Modo 1 Imagem Foi Selecionado*

O framework adiciona na cena o *prefab ImageTarget* do Vuforia em seguida adiciona na árvore do *ImageTarget* o modelo 3D com o nome igual ao informado pelo desenvolvedor na interface de usuário, se o modelo 3D não ficar na mesma árvore do *ImageTarget* ao fazer o rastreamento da carta o modelo não será exibido.

Depois o sistema realiza uma chamada ao ***Módulo Geração da Base*** para criar ou aumentar a base de dados das cartas e, com base no endereço da imagem fornecida pelo desenvolvedor o framework identifica o nome da imagem e o endereço da pasta onde ela está salva passando essas informações na chamada do ***Módulo de Obtenção dos Atributos de Uma Carta*** e encerra o módulo.

2ºCaso: *Se Modo Várias Imagens Foi Selecionado*

O framework obtém uma imagem da pasta cujo endereço foi fornecido pelo desenvolvedor, ler o arquivo de atributos com o mesmo nome da imagem para descobrir o nome do modelo 3D que será utilizado para a carta. Adiciona na cena o *GameObject ImageTarget* e em sua árvore adiciona o modelo 3D.

Em seguida é feita uma chamada ao ***Módulo de Geração da Base*** e outra chamada é feita ao ***Módulo de Geração de Scripts*** para criar os scripts específicos.

Caso ainda exista imagem na pasta que o sistema gerou um *GameObject* o framework retorna ao passo de "Obter imagem da pasta", se todas as imagens já foram processadas o framework encerra o módulo.

3.2.2 Módulo de Interface Configuração

Responsável pela criação do arquivo de configuração, composto pela classe **Primeiro_Acesso** (Figura 8 exemplifica interface deste módulo).

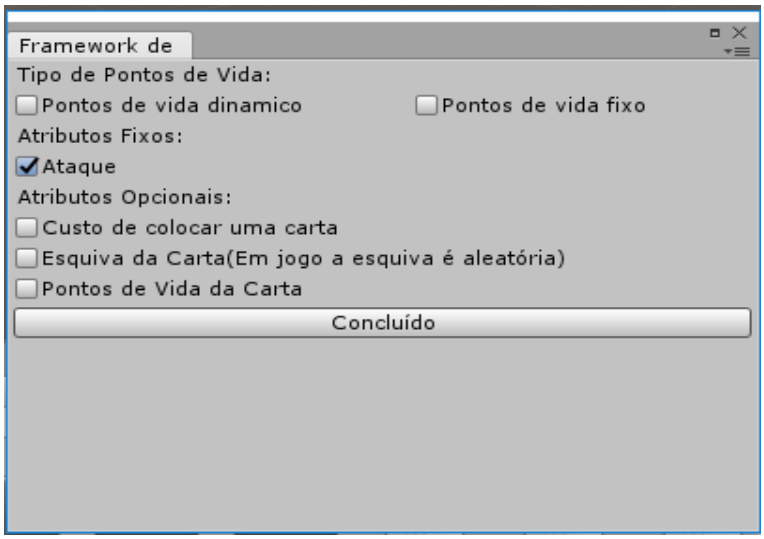

Figura 8 – Interface Configuração.

O objetivo deste módulo é fornecer ao desenvolvedor as opções do sistema de jogabilidade e facilitar a coleta dessas informações. A Figura 9 traz a visão de como são executadas as etapas de processamento deste módulo.

Quando este módulo é chamado o framework abre automaticamente uma janela de interface. O desenvolvedor então seleciona quais das opções ele deseja que esteja presente no seu jogo (todas as opções possíveis serão descritas na próxima seção).

Se clicar em fechar o framework encerra, se clicar no botão Concluído o framework obtém os dados de quais elementos foram selecionados e uma pasta com o nome **Config_Jogo** e dentro dela cria o arquivo de configuração com o nome de **Config**, no **Asset Database** e encerra o módulo voltando o controle do sistema ao módulo que o chamou.

3.2.3 Módulo de Obtenção dos Atributos de Uma Carta

Responsável por intermediar a coleta de informações referentes à uma carta e criar o arquivo de atributos referente a esta carta.

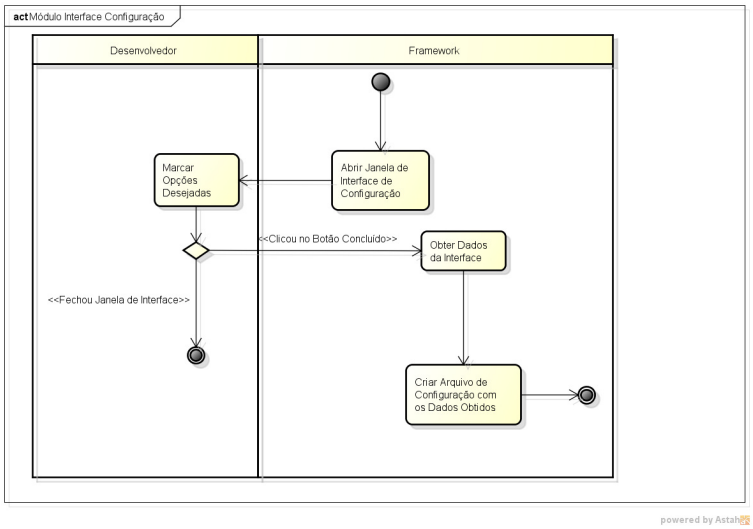

Figura 9 – Diagrama do Atividades do Módulo Interface Configuração.

Este módulo possui uma interface personalizada que é alterada de acordo com os elementos escolhidos no ***Módulo de Interface de Configuração***, através desta interface o usuário entra com os valores dos atributos da carta. A Figura 10 exemplifica a interface do módulo.

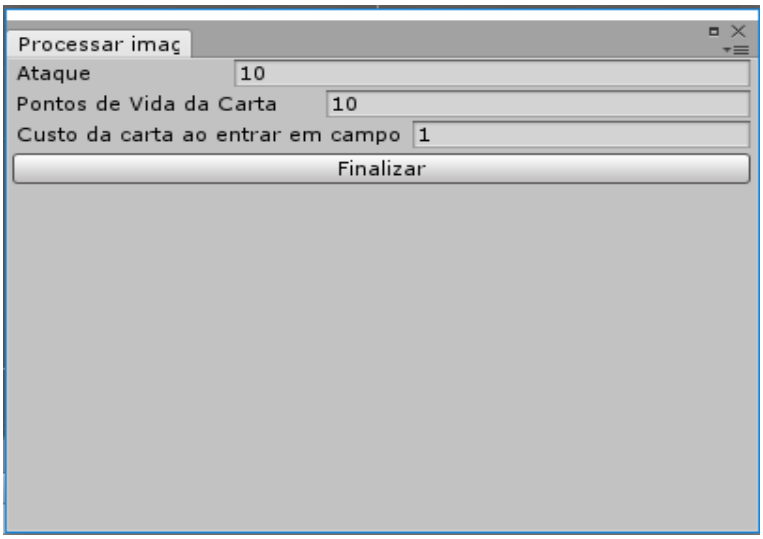

Figura 10 – Exemplo de Interface para Obtenção dos Atributos de Uma Carta.

Este módulo é chamado apenas no modo "1 Imagem"e seu funcionamento, descrito na Figura 11, é detalhado abaixo:

Quando este módulo é iniciado o framework realiza uma leitura no arquivo de configuração para ver quais elementos o desenvolvedor escolheu para estar presente no jogo, depois configura a interface do usuário com os elementos presentes no arquivo de configuração e que são referentes à carta. Após configurada a interface o framework a exibe pro desenvolvedor e aguarda sua ação.

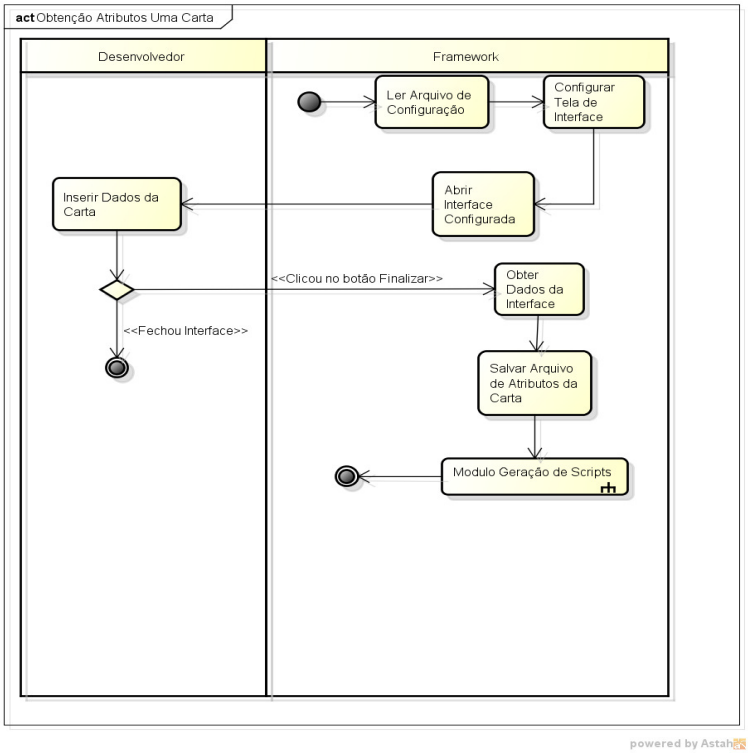

Figura 11 – Diagrama Atividades do Módulo de Obtenção dos Atributos de Uma Carta.

O desenvolvedor insere as informações da carta nos campos específicos e se clicou no botão Finalizar o framework obtém os dados informados e salva as informações no arquivo de atributos, que é salvo com o nome e no endereço fornecido pelo módulo que realizou a chamada.

Depois o framework realiza uma chamada ao ***Módulo Geração de Scripts*** para criar os scripts específicos da carta e encerra, retornando ao módulo que o chamou.

3.2.4 Módulo Geração da Base

Este módulo é responsável por gerenciar a comunicação com a biblioteca Vuforia para criar a base de rastreamento das imagens, importar para o jogo, definir a base que será utilizada no jogo e referenciar os *ImageTargets* na cena do jogo com os dados de rastreamento da imagem referente a carta, que o *ARCamera* usará para comparar com as cenas capturadas do mundo real. Abaixo estão detalhadas as atividades executadas neste módulo e que foram descritas na Figura 12.

Abaixo está detalhado as atividades executadas neste módulo e que foram descritas na Figura 12.

Ao ser iniciado, este módulo estabelece uma comunicação com o Vuforia em seguida verifica se o desenvolvedor já possui uma base pro jogo, procurando por uma base com o mesmo nome do jogo. Se não encontrar nenhuma uma nova base, com o mesmo nome do

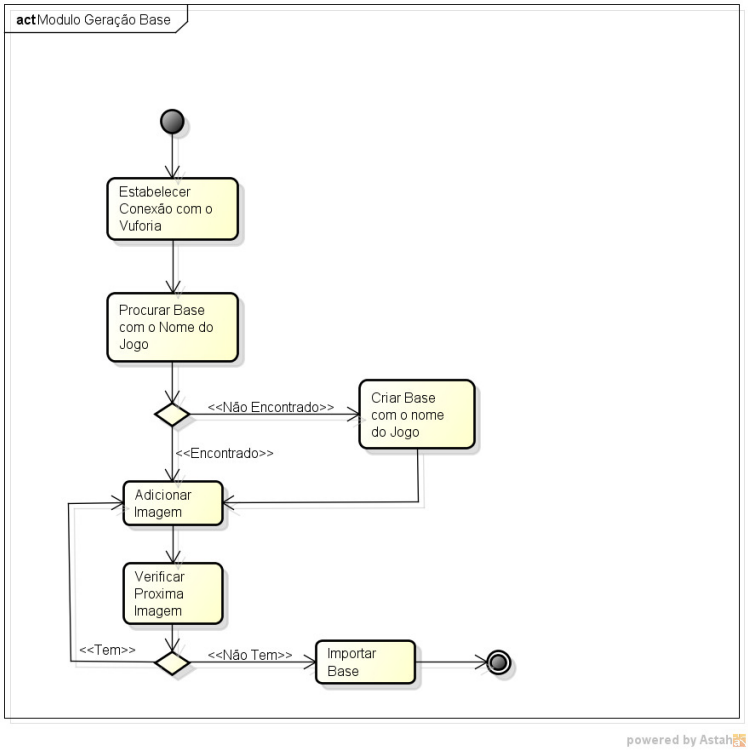

Figura 12 – Diagrama Atividades do Módulo Geração da Base.

jogo, é criada.

Caso encontre a base adiciona a imagem na base e verifica se existem outras imagens a serem adicionadas, se existir retorna ao passo de adicionar imagem senão importa a base para o jogo e encerra.

3.2.5 Módulo Geração de Scripts

O objetivo deste módulo é gerar todos os scripts necessários para a execução do jogo, não é responsabilidade dele a atribuição dos scripts aos *GameObjects* instanciados na cena do jogo. Está dividido em duas classes: Criação de Scripts Específicos e Criação de Scripts Gerais.

A classe de criação de scripts específicos cria os arquivos com base num protótipo e altera a informação que é específica de cada carta, como nome, valor de ataque e pontos de vida. Neste caso, o arquivo padrão sofre leves modificações afim de corresponder a carta ao qual o script está sendo gerado.

Já na criação dos Scripts Gerais os códigos não sofrem alterações e quando adicionado a um GameObject a Unity instancia um clone dos scripts. Esta classe é chamada apenas uma vez, mesmo que o desenvolvedor tenha optado por trabalhar com várias imagens.

Já na criação dos Scripts Gerais os códigos não sofrem alterações e quando adicio-

nado a um GameObject a Unity instancia um clone dos scripts. Esta classe é chamada apenas uma vez, mesmo que o desenvolvedor tenha optado por trabalhar com várias imagens. Neste módulo foram utilizados os Padrões de Criação *Prototype* e *Singleton*.

O Padrão Prototype utilizado como referência para melhorar a criação dos scripts de jogo, tanto os específicos quanto os gerais. Nos scripts gerais foi criado o protótipo dentro do próprio código, essa foi uma estratégia para caso no futuro esses scripts passem a ter comportamento de scripts específicos, facilitando essa migração.

Nos scripts específicos é adicionado elementos que são exclusivos de cada carta, como o nome e valor de ataque, perceba que são os valores que modificam e não a variável em si. Assim, utilizando a ideia do *Prototype* criamos um protótipo em linha de código onde é alterado a inserção dos valores específicos nas variáveis.

Já o Padrão de Projeto *Singleton* é utilizado para limitar o acesso a criação dos scripts, ele limita todas as demais classes que fazem chamada a criação dos scripts de jogo a usarem um único método público (um método público para criar scripts específicos e um método público para criar scripts gerais, já que são classes diferentes), isso garante que todas as informações necessárias para criar os scripts de jogo serão fornecidas e que todos os scripts necessários para o *GameObject* serão de fato criados.

A figura 13 mostra o diagrama de atividades deste módulo, abaixo é apresentada a descrição de cada atividade.

Quando o módulo é iniciado o framework verifica se foi chamado para criação de scripts gerais ou se foi para criação de scripts específicos, como são duas classes distintas que tratam da criação de cada caso, temos dois fluxos de execução diferentes a descrição das atividades seguirão a ordem em que as atividades acontecem.

Caso Solicitado a Criação de Scripts Gerais

O framework cria o script de base dos elementos que representam as características das cartas; cria o script de controle das animações, responsável por controlar as animações e alterações de valores na máquina de estados (explicada no ***Módulo Geração de Controle Animações***); Cria o scrip de Gerenciamento dos jogadores, arquivo que controla as informações referentes aos jogadores(como turno e pontos de vida); e cria os scripts de comportamento das cartas, que controla as ações e informações das cartas.

Depois salva os arquivos no *Asset Database* da Unity para que possam ser utilizados no jogo e atualiza o *Asset Database*, senão os arquivos não ficarão visíveis e não poderão ser usados e encerra o módulo.

Caso Solicitado a Criação de Scripts Específicos

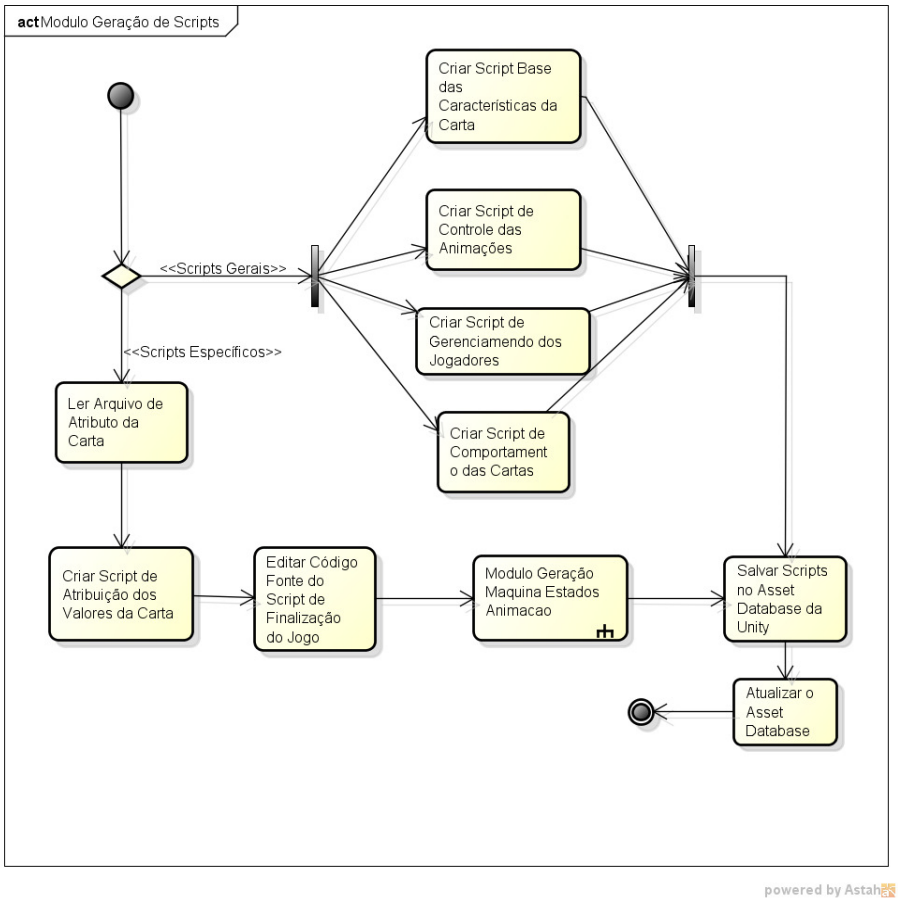

Figura 13 – Diagrama Atividades do Módulo de Geração de Scripts.

Quando é solicitado a este módulo a criação dos scripts específicos o framework lê o arquivo de atributos da carta para obter os dados da carta e em seguida adiciona esse valores no script especifico da carta para cada vez que o jogo iniciar esse valores sejam carregados, mantendo as características individuais de cada carta.

Em seguida, é adicionado a informação no código da classe que adiciona os elementos criados aos *GameObjects* na cena em qual *GameObject* o script deve ser inserido e então realiza uma chamada ao ***Módulo Geração de Controle Animações***, depois salva os scripts criados no *Asset Database* e o atualiza, encerrando assim o módulo.

3.2.6 Módulo Geração de Controle Animações

As animações na Unity precisam de uma máquina de estados que controla a transição de uma animação pra outra que define qual animação cada estado poderá rodar e em que condições as transições devem ocorrer. Um exemplo disso é quando o *GameObject* precisa alterar da animação *Parado* para a animação *Atacando*, nesse caso a Unity verifica se é possível realizar a transição de um estado para outro (Uma seta direcionada de um estado para outro) e as condições para a transição (na imagem o parâmetro *Parado* tem

valor falso e o parâmetro *Atacando* tem valor verdadeiro). A Figura 14 mostra um exemplo de máquina de estados.

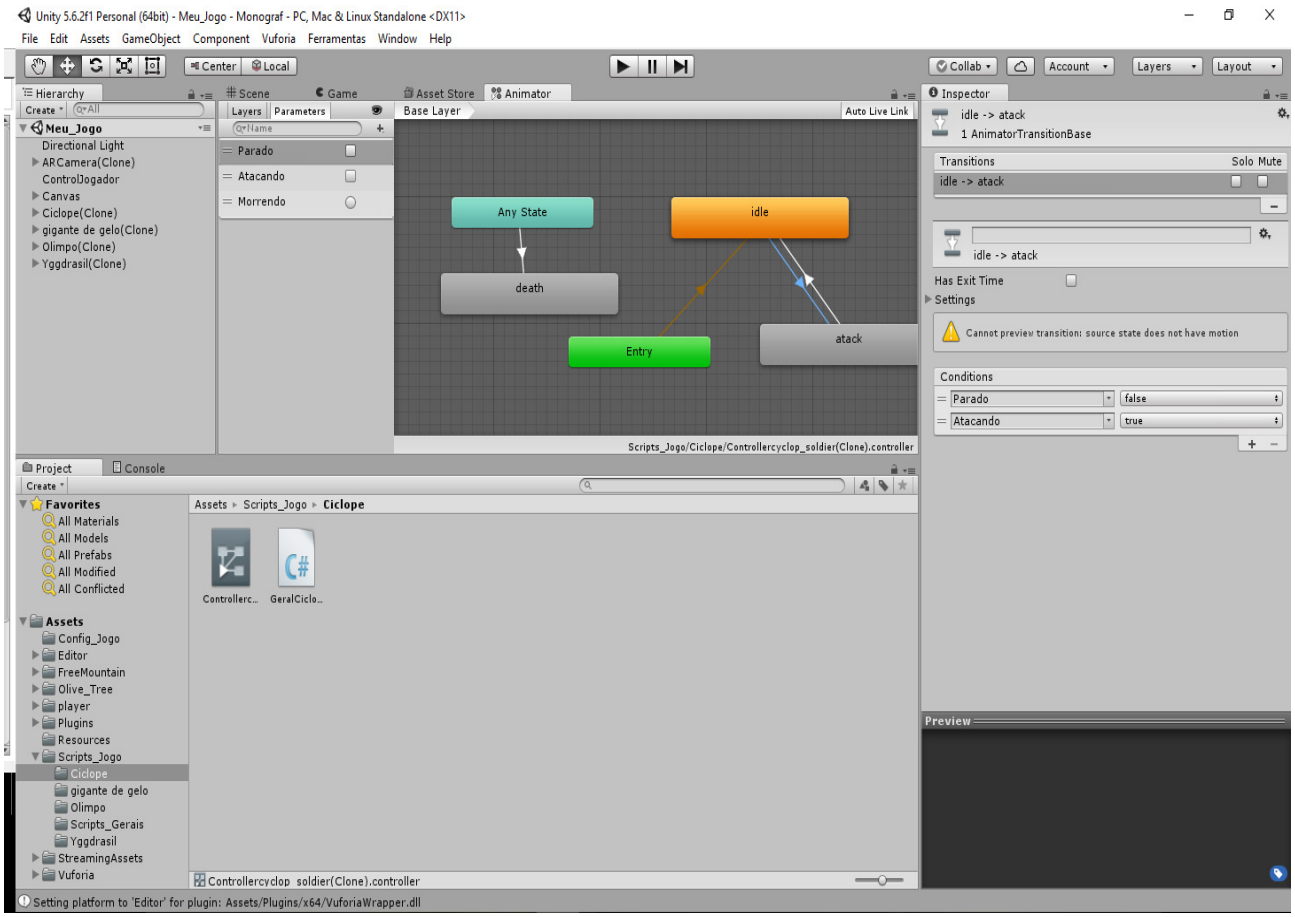

Figura 14 – Máquina de Estados para Controlar Animação.

Levando em consideração esse caso, foi utilizado um módulo específico para criação da máquina de estados. Neste módulo é criado a máquina de estados, seus parâmetros, os estados de cada elemento e as condições de cada transição. A Figura 15 apresenta as etapas de execução para gerar a máquina de estados.

Ao ser iniciado cria uma nova máquina de estados na pasta dos scripts específicos. Após a máquina de estados criada deverá ser preenchida, a primeira coisa a se inserir são os estados (eles irão conter as animações que serão executadas quando o estado estiver ativo), no exemplo da Figura 14 são representados pelos retângulos.

Depois dos estados criados são inseridos as transições de estados, indicadas por setas (as transições limitam a mudança de estados, elas indicam, que dado um estado, para qual outro estado é possível ir a partir dele). Em seguida são adicionados as condições para que seja possível realizar uma transição de um estado para outro.

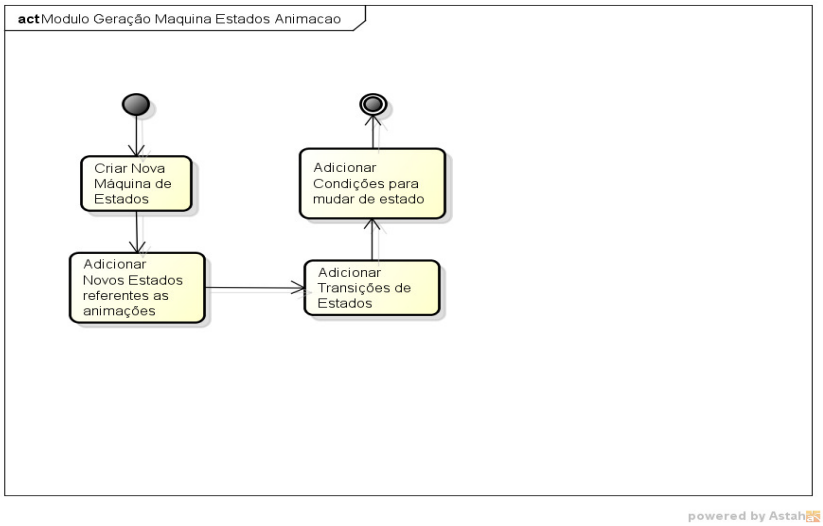

Figura 15 – Diagrama de Atividades do Módulo Geração de Controle Animações.

3.2.7 Interface de Finalização do Jogo

Neste módulo todos os scripts e controles de animação são adicionados aos respectivos GameObjects na cena. Este módulo é composto por duas classes (**addElementos** e **addElementoL**).

Este módulo precisou ser separado dos demais porque os scripts do jogo são gerados em tempo de execução, só podem ser instanciados na cena após os arquivos geradores serem finalizados e a *Asset Database* atualizada (esta atualização ocorre no processo de encerramento dos código que geram os scripts). Em caso dos scripts não tiverem sido gerados e o código responsável pelas instanciações desses scripts esteja em um arquivo que não esteja sendo executado, ainda sim a Unity acusará um erro e não permitirá a execução dos demais scripts ou não irá rodar os códigos de forma correta.

A Figura 16 retrata as atividades executadas por este módulo e abaixo essas atividades são detalhadas.

Quando o usuário escolhe a opção Finalizar jogo no menu ADGAR o framework abre uma janela de interface (Figura 17) para que o desenvolvedor confirme se deseja finalizar o jogo, para isso basta ele clicar no botão "Encerrar".

Depois do desenvolvedor clicar no botão o framework passa para a etapa de adicionar os scripts criados ao GameObjects presentes na cena do jogo. Depois de todos os elementos adicionados os códigos que foram inseridos na classe são apagados e o módulo encerra.

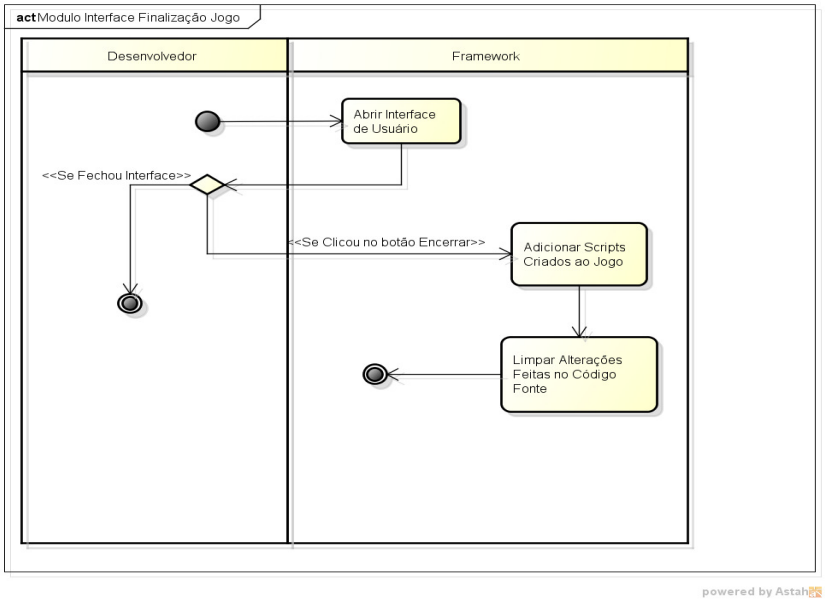

Figura 16 – Diagrama de Atividades do Módulo de Finalização do Jogo

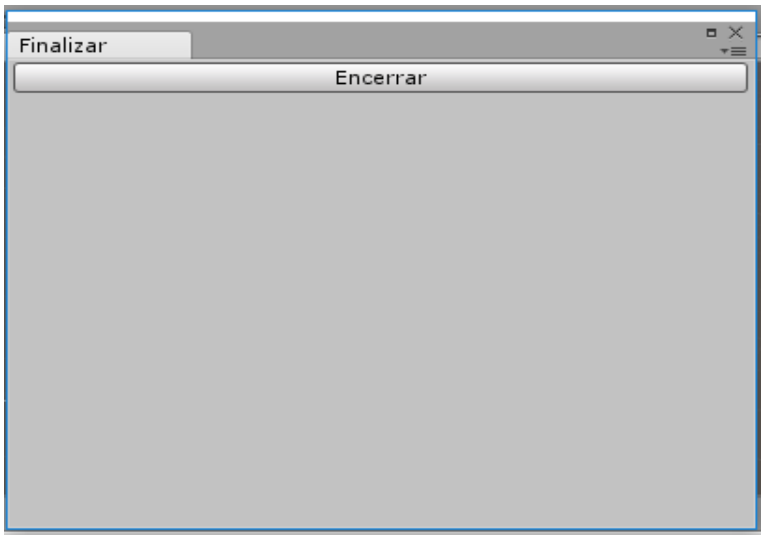

Figura 17 – Interface de Finalização do Jogo

4 Estudo de Caso: Apocalipse Mitológico Cartas Colecionáveis

O jogo Apocalipse Mitológico foi modelado como um meio de testar o ADGAR Framework. Sua temática é sobre elementos de várias mitologias. Um jogador enfrenta o outro e o objetivo é deixar os pontos de vida do jogador adversário com menos de zero pontos. O jogo possui pontos de vida do jogador do tipo dinâmico, pontos de ataque da carta e pontos de vida da carta.

Há dois tipos de carta no jogo, as cartas de vida, que aumentam os pontos de vida do jogador que as jogam e pontos de ataque igual a zero; e as cartas de monstros que são usadas para atacar o jogador adversário.

Com as opções escolhidas o pontos de vida de cada jogador inicialmente é zero e aumenta de acordo com as cartas de vida jogadas, cada jogador inicia com uma carta de vida na mão e elas tem custo de jogada zero. Na primeira rodada uma carta de vida deve ser jogada (se uma carta diferente for jogada os pontos de vida do jogador ficarão menor que zero e este perde). Os danos nos pontos de vida do jogador acontecem quando é zerado os pontos de vida carta. Se um jogador não tem cartas de monstros não pode atacar e o jogador adversário pode atacar diretamente os seus pontos de vida diretamente.

Na primeira rodada não é possível atacar pois, como mostrado, as primeiras cartas a serem jogadas são as de vida que possuem pontos de ataque igual a zero. Os pontos de vida dos jogadores devem ser mostrados na tela de visão do usuário, ter animações parado e de ataque, o modelo 3D deve sumir quando a carta morre.

4.1 Construindo o jogo no ADGAR Framework

1. Ao abrir o Unity3D foi criado o projeto do jogo, após a engine abrir a pasta de desenvolvimento foi instalado o Vuforia no Unity e importado os modelos 3D para atender aos requisitos de funcionamento do framework. Foi utilizado a versão 6.2 do Vuforia.

2. Ao abrir a tela principal no menu ADGAR, que aparece juntos dos menus da Unity, o sistema encaminha para a tela de configurações, foram marcadas as opções que se encaixavam com a proposta do jogo, como mostra a Figura 18.

3. Após finalizado a escolha das configurações o framework apresentou a tela principal, nesta interface optou-se por trabalhar no modo várias imagens e fornecido o endereço da pasta onde estão salvas as imagens e os arquivos de atributos (Figura 19).

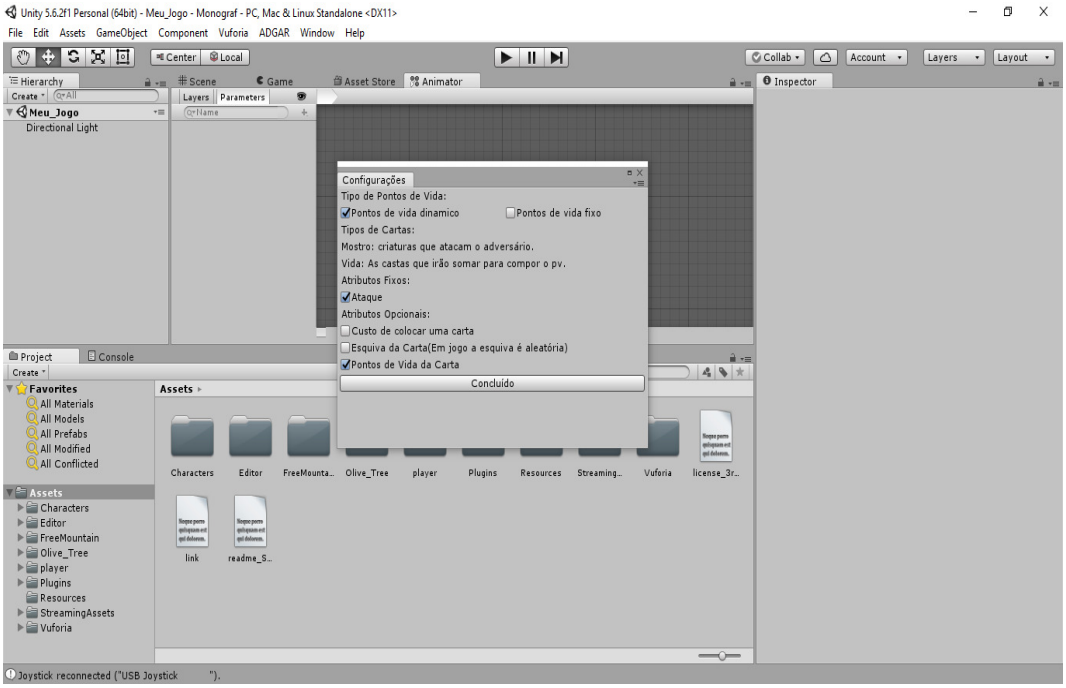

Figura 18 – Configurações escolhidas no desenvolvimento do jogo Apocalipse Mitológico

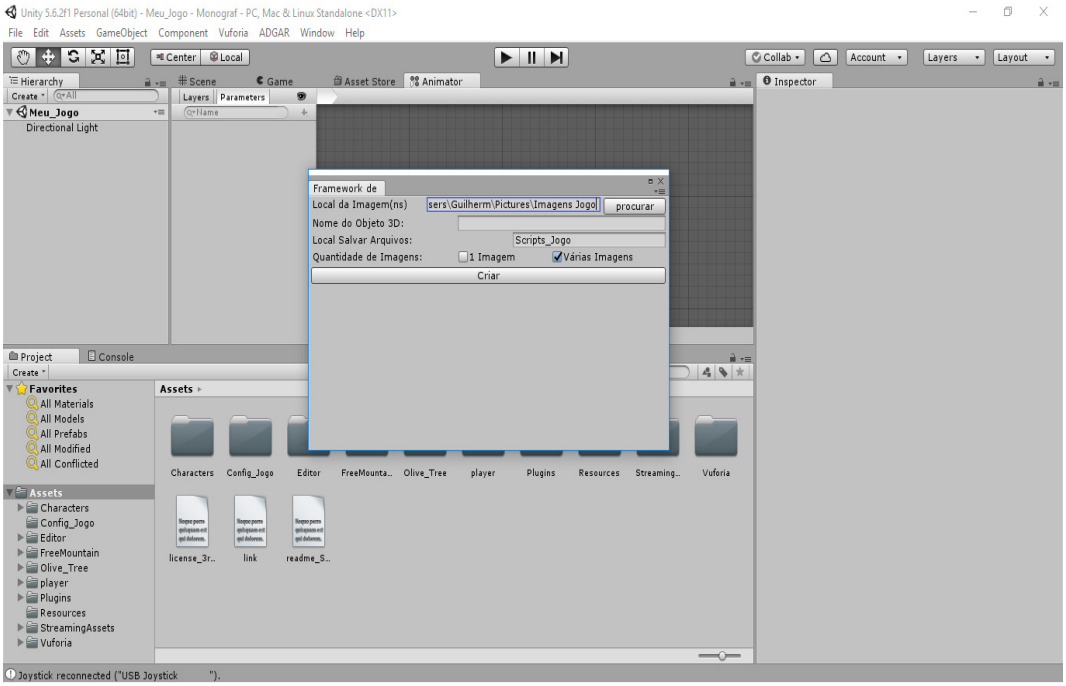

Figura 19 – Tela principal pro desenvolvimento do jogo Apocalipse Mitológico

Depois da conclusão desta etapa já temos os elementos de jogo e os scripts criados marcados com o retângulo vermelho na Figura 20.

4. Neste passo novamente é acessado o menu ADGAR e escolhido a opção "Finalizar Jogo"que exibe a tela mostrada na Figura 21.

5. Por fim é adicionado as animações à máquina de estados dos GameObjects das cartas

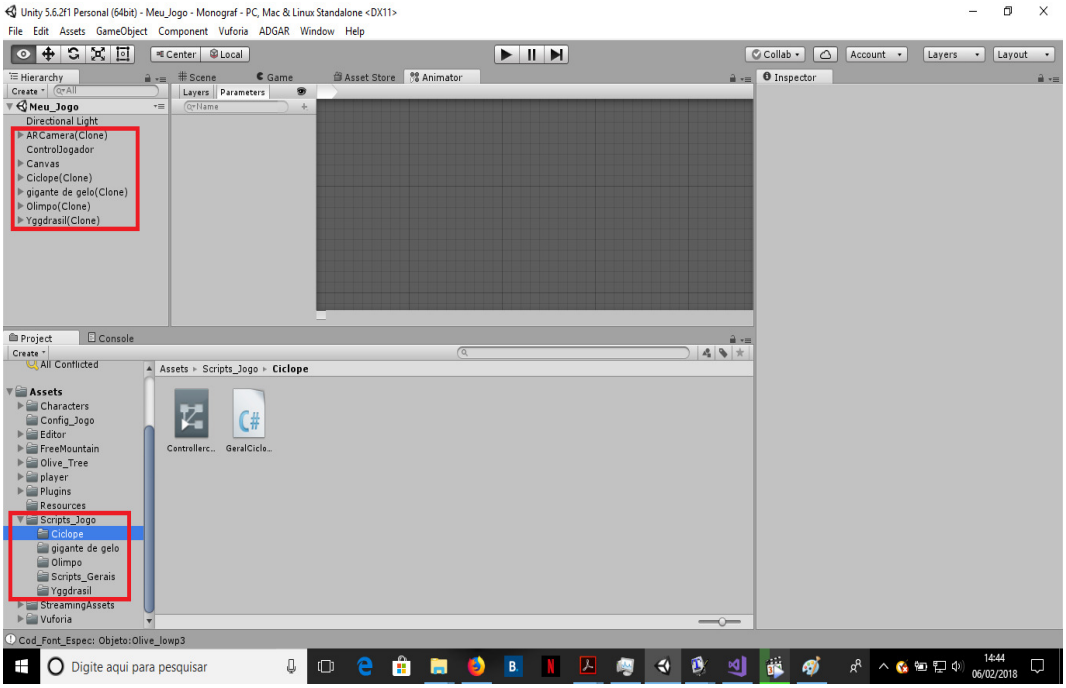

Figura 20 – Elementos criados e GameObjects adicionados à cena do jogo Apocalipse Mitológico

mostrado na Figura 22 pela seta vermelha.

4.2 Resultados

O framework implementou com sucesso os componentes de interface dos pontos de vida de cada jogador, sendo exibido os pontos de vida inicial (Figura 23), aumentando eles quando uma carta de vida é jogada (Figura 24), decrementa o custo de uma carta da vida do jogador quando uma carta de monstro é jogada (Figuras 25 e 26) e quando os pontos de vida da carta são zerados o valor é diminuído dos pontos de vida do respectivo jogador(Figura 27).

O deslocamento do modelo 3D da carta que ataca é feito com sucesso (Figura 27), a animação de ataque é executada, embora em alguns casos com um leve erro de sincronismo, fazendo com que a carta retorne a posição inicial antes da animação de ataque terminar.

Quando uma carta morre o modelo 3D some de acordo com o planejado, embora nenhuma animação seja executada pra representar a morte da carta.

Os GameObjects e Scripts de jogo são criados automaticamente com sucesso e executam sem erro. Embora o framework agilize o desenvolvimento do jogo, algumas configurações são ainda feitas manualmente, como a adição das animações aos respectivos estados, as configurações do Vuforia (número de modelos renderizados simultaneamente no arquivo de configuração do próprio Vuforia) e a base teve de ser criada manualmente.

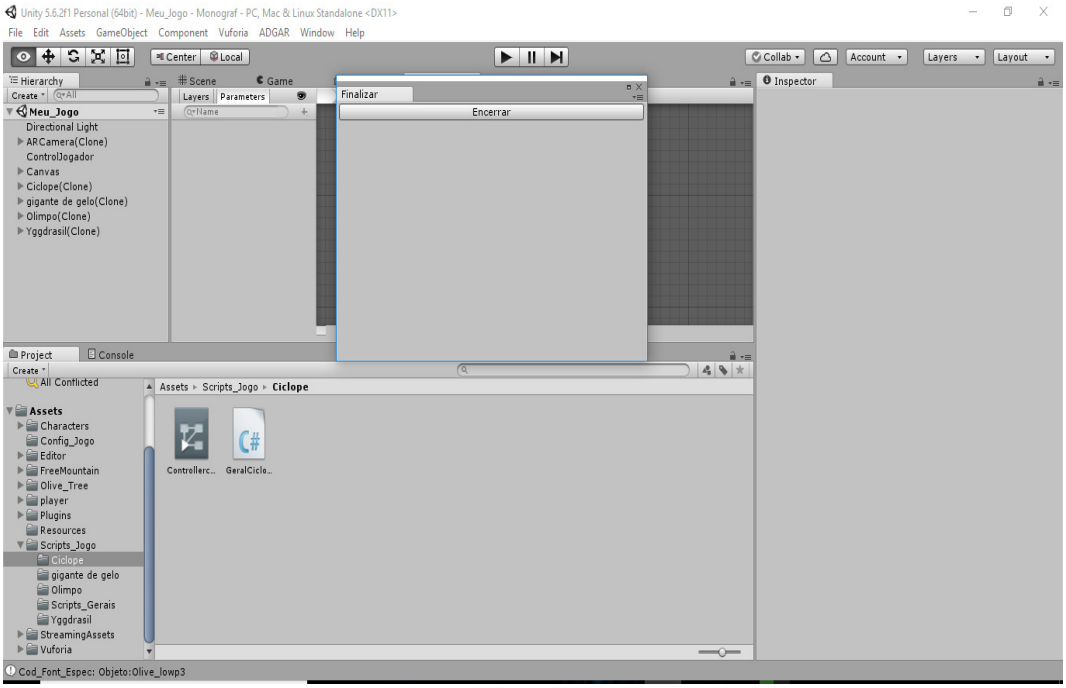

Figura 21 – Interface de Finalização do jogo

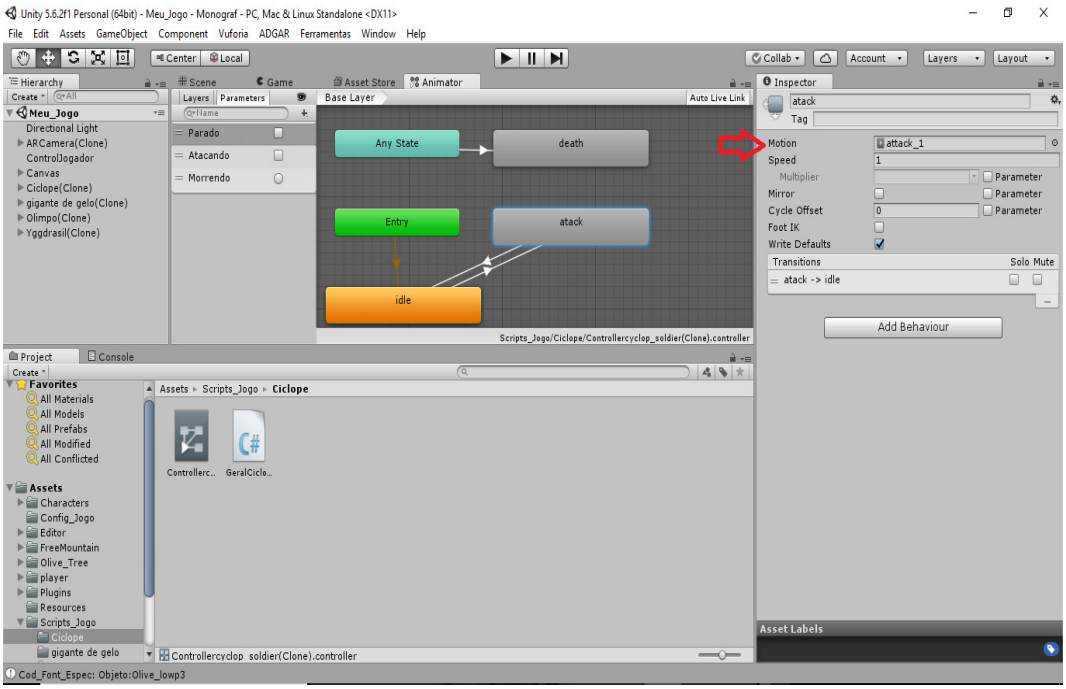

Figura 22 – Máquina de Estados das Animações de uma Carta

Apesar de criada de forma manual não comprometeu os elementos de jogo já criados.

Os pontos de vida e ataque das cartas não aparecem na tela do jogo (como é possível ver nas Figuras 25 e 27), o que prejudica um pouco o planejamento de estratégia durantes as partidas jogadas.

O erro de sincronismo da animação de ataque pode ser contornado criando uma

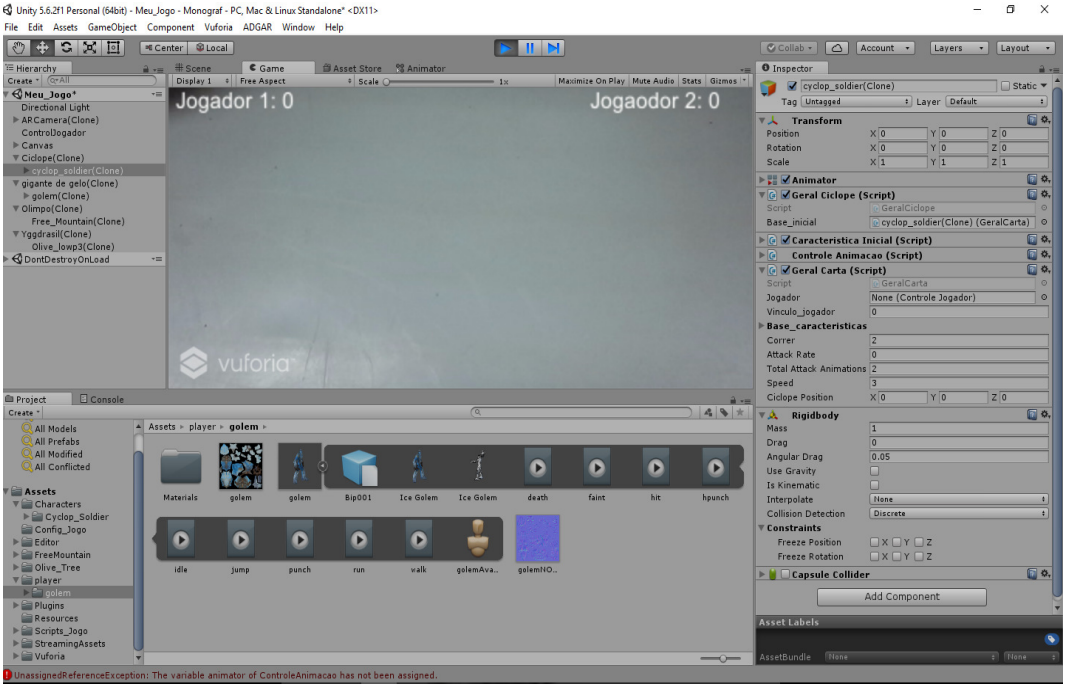

Figura 23 – Pontos de Vida Inicial do Jogo

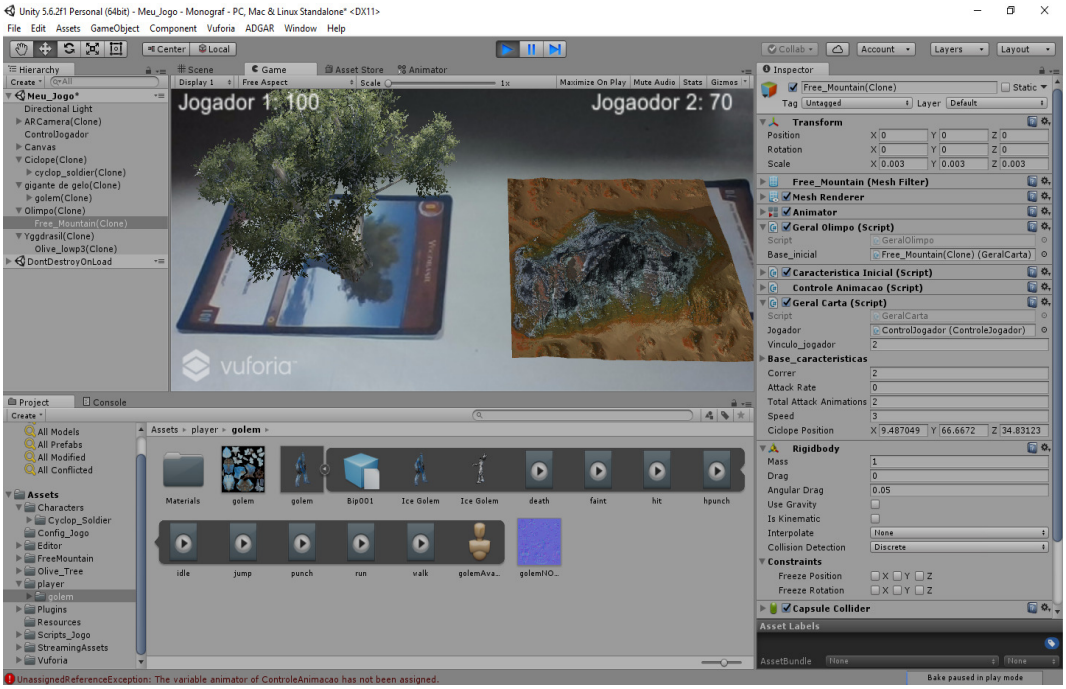

Figura 24 – Pontos de Vida quando jogado uma carta de vida

função de ajuste do tempo, as animações não tem tempo de execução iguais logo, essa função tem que ser dinâmica de modo que se ajuste a cada tempo de animação dos modelos das cartas.

Para corrigir a falta de informação dos pontos de vida e ataque da carta é possível adicionar dentro da classe de criação de scripts gerais, na classe que controla os dados

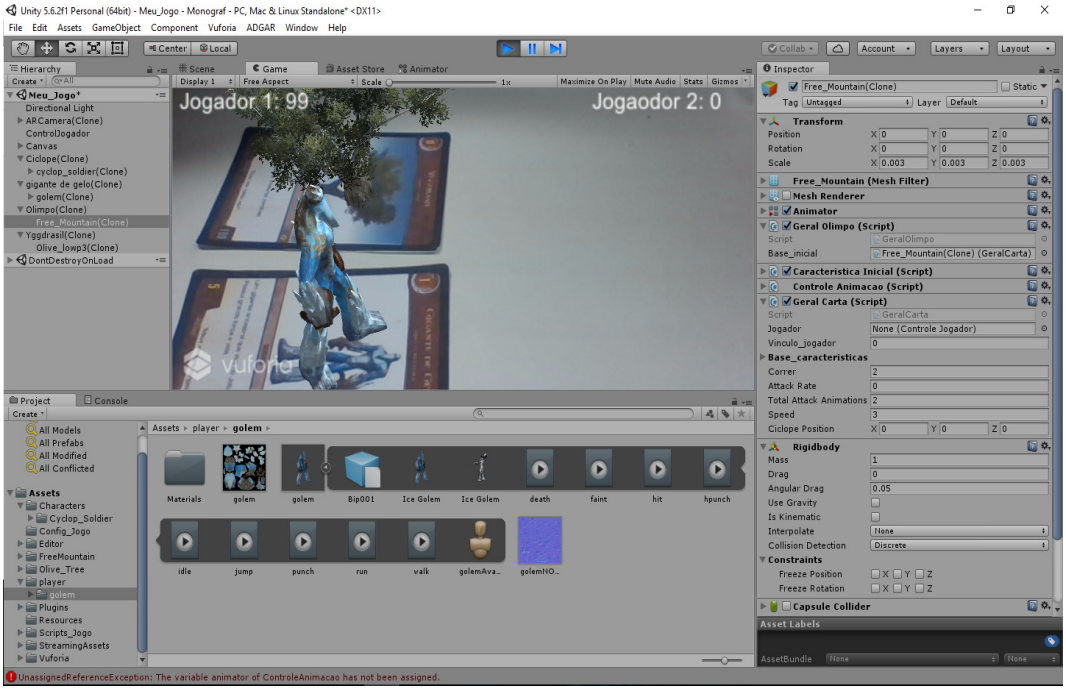

Figura 25 – Jogando uma carta de monstro

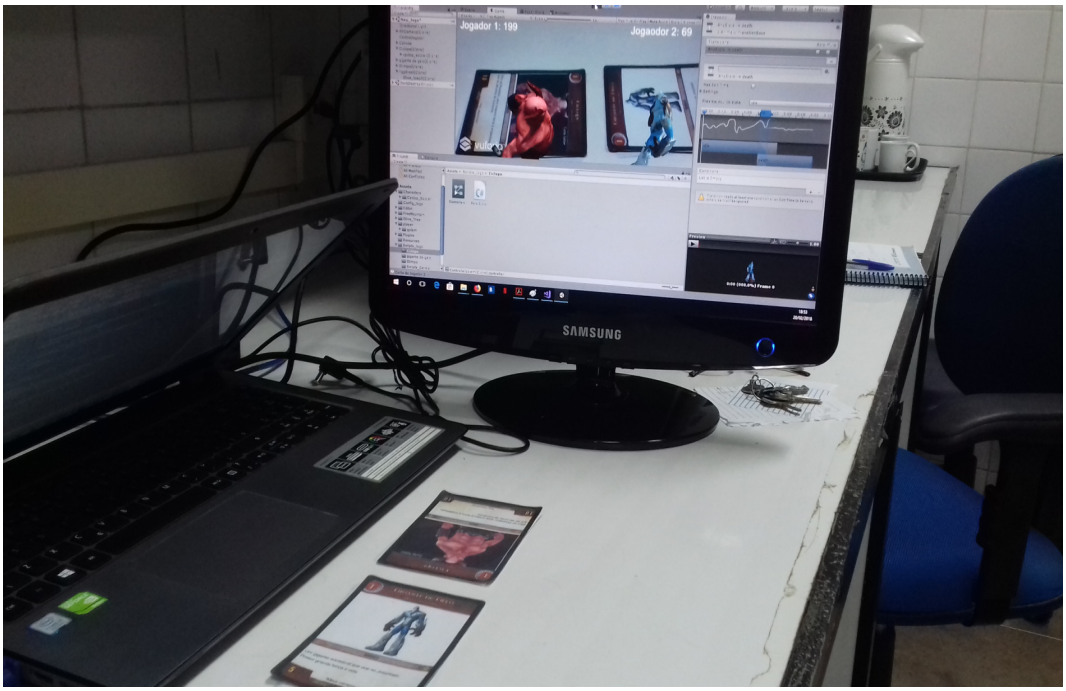

Figura 26 – Dois jogadores com uma carta cada um

gerais de uma carta, um elemento de interface de texto anexado a carta, sendo exibido junto quando o objeto da carta aparece e sumido junto também.

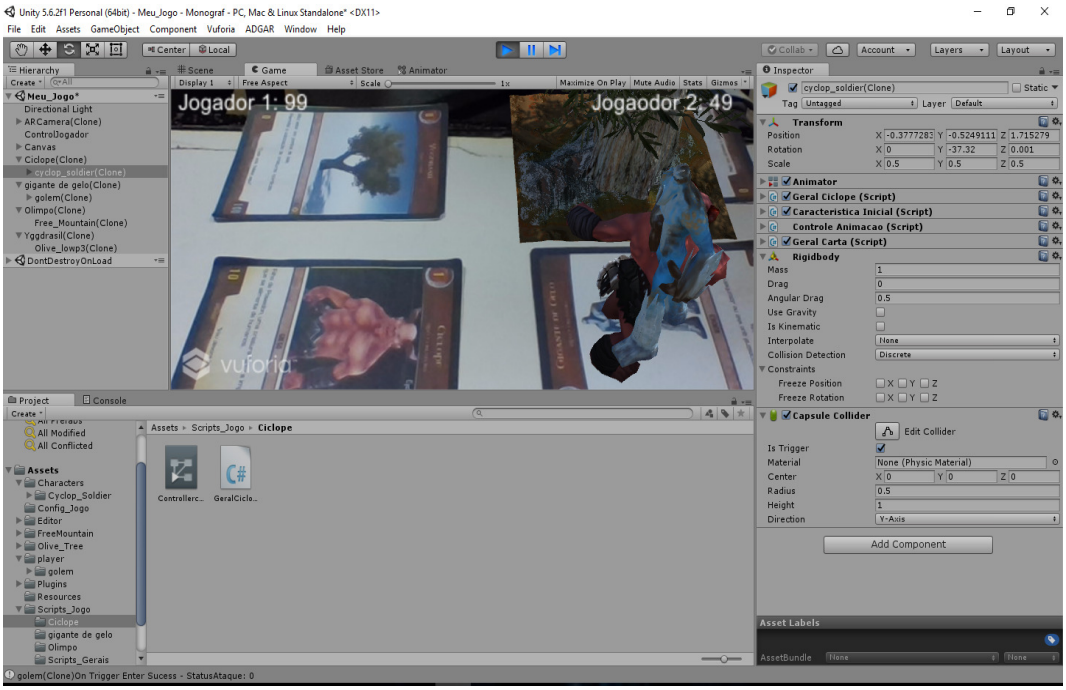

Figura 27 – carta Atacando

5 Conclusão

Este trabalho propôs uma forma de agilizar e minimizar os erros no desenvolvimento de jogos com RA através do desenvolvimento de uma ferramenta, o ADGAR Framework, que se vale do reuso de código para criar GameObjects e scripts automaticamente na cena do jogo.

A parte de automatização na geração de scripts do jogo foi alcançada com sucesso, todos os scripts necessários para o funcionamento do jogo foram criados e rodam sem erros. Os GameObjects também são criados na cena do jogo sem problemas, embora nas mesmas posições, o que não prejudica o funcionamento do jogo mas dificulta a visualização do desenvolvedor de todos os elementos do jogo.

A ferramenta teve também como proposta abstrair a parte de programação do desenvolvedor do jogo.

O trabalho, com o estudo de caso, agilizou o processo de desenvolvimento de jogo e abstraiu as partes de programação do jogo. Contudo a atribuição da animações de forma manual por parte do desenvolvedor ainda deixa relativamente lento o processo de desenvolvimento do jogo.

5.1 Trabalhos Futuros

Como trabalhos futuros, sugerimos algumas melhorias no framework:

- Manutenção do módulo para abstração dos componentes do Vuforia para automatizar a criação da base e configurações do Vuforia;
- Desenvolvimento de uma abordagem para padronização das animações utilizadas no jogo para que o framework as instancie automaticamente durante a execução do ***Módulo de Criação de Scripts***;
- Melhorar o sincronismo das animações através de uma classe que gerencie melhor o tempo de execução delas.
- Integrar o proposto por (MIYAZAKI; REY; MARENGONI, 2012) no jogo como uma proposta de jogo imersivo, aumentando as opções de jogabilidade. A ideia é projetar os modelos 3D das cartas na mesa onde as cartas estão sendo jogadas.

Referências

ARTOOLKIT. 2004. Disponível em: <www.hitl.washington.edu/artoolkit>. Acesso em: 06 de jan 2018.

BOTEGA, L. C.; CRUVINEL, P. E. *Realidade Virtual: Histórico, Conceitos e Dispositivos*. Porto Alegre - RS, Brasil, 2009. 08-30 p.

BRIZENO, M. *Padrões de Projeto*. 2011. Disponível em: <https://brizeno.wordpress.com/padroes/>. Acesso em: 11 de jan de 2018.

BRIZENO, M. *Primeiros passos com Padrões de Projeto*. [S.l.]: Leanpub, 2016.

BRIZENO, M. *Refatorando com padrões de projeto Um guia em Ruby*. [S.l.]: Casa do Código, 2016.

CAROLINE, E. *Jogos de Cartas Online*. 2017. Disponível em: <https://medium.com/tendências-digitais/a-ascens~ao-dos-jogos-de-carta-online-44ff6be85c36>. Acesso em: 02 jan 2018.

COPAG. *TUDO SOBRE BARALHOS - ORIGENS*. 2018. Disponível em: <http://copag.com.br/tudo-sobre-baralhos/origens/>. Acesso em: 02 jan 2018.

GALAM, G. T.; DIAS, M. A. Realidade aumentada aplicada a um jogo de cartas baseado em rpg. In: XVI SIMPóSIO BRASILEIRO DE JOGOS E ENTRETENIMENTO DIGITAL. 2017. Disponível em: <https://www.sbgames.org/sbgames2017/papers/ARTES_E_DESIGN/SHORT_PAPERS/175323_2_versao_preliminar.pdf>. Acesso em: 09 de jan de 2018.

GARFIELD, R. et al. *Manual de Regra Basico*. Wizards of the Coast LLC, 2013. Disponível em: <https://wpn.wizards.com/sites/wpn/files/attachements/pt_mtgm14_rulebook_web.pdf>. Acesso em: 02 jan 2018.

JACINTHO, B. P. *O QUE É O TRADING CARD GAME*. 2013. Disponível em: <https://estruturaludens.wordpress.com/2013/07/03/o-que-e-o-trading-card-game/>. Acesso em: 02 jan 2018.

KIRNER, C.; TORI, R. *Fundamentos de Realidade Aumentada*. Belém - PA, Brasil, 2006. 22-38 p.

MADEIRA, C. A. G. *FORGE V8: Um framework para o desenvolvimento de jogos de computador e aplicações multimídia*. Recife - PE: [s.n.], 2001.

MALHEIROS, T.; REIS, L.; CARVALHO, J. E. R. de. A utilização da realidade aumentada em jogos de cartas colecionáveis. In: XI SBGAMES. Brasília - DF, 2012.

MARON, G.; MARSON, F. *C.A.V.E.I.R.A.: A Framework for Developing Augmented Reality Card Games*. 2009. Disponível em: <http://www.lbd.dcc.ufmg.br/colecoes/svr/2009/0021.pdf>. Acesso em: 02 jan 2018.

MELO, R. *Grupo da Bahia desenvolve card game sobre mitos com tecnologia 3D*. 2015. Disponível em: <http://g1.globo.com/bahia/game-bahia/2015/noticia/2015/05/grupo-da-bahia-desenvolve-card-game-sobre-mitos-com-tecnologia-3d.html>. Acesso em: 22 de fev de 2018.

MIYAZAKI, A. K.; REY, G. P.; MARENGONI, M. *Biblioteca Digital Brasileira de Computação*, 2012. Disponível em: <http://www.lbd.dcc.ufmg.br/bdbcomp/servlet/Trabalho?id=20996>. Acesso em: 15 de dez de 2017.

OSGART NG. *Overview of osgART*. 2009. Disponível em: <http://www.osgart.org/index.php/Overview_of_osgART>. Acesso em: 02 jan 2018.

PANINI. 2017.

SILVEIRA, D. *Número de desenvolvedores de games cresce 600% em 8 anos, diz associação*. 2017. Disponível em: <https://g1.globo.com/economia/negocios/noticia/numero-de-desenvolvedores-de-games-cresce-600-em-8-anos-diz-associacao.ghtml>. Acesso em: 22 de fev de 2018.

SONY. 2007. Disponível em: <https://www.playstation.com/pt-pt/games/the-eye-of-judgment-ps3/>.

TORI, R. et al. *Jogos e Entretenimento com Realidade Virtual e Aumentada*. Petrópolis - RJ, Brasil: Editora SBC, 2007. 192-222 p.

UNITY. 2017. Disponível em: <https://unity3d.com>. Acesso em: 15 dez 2017.

VINHA, F. *Conheça os jogos com realidade aumentada similares a Pokemon Go*. 2016. Disponível em: <http://www.techtudo.com.br/listas/noticia/2016/07/conheca-os-jogos-com-realidade-aumentada-similares-pokemon-go.html>. Acesso em: 06 de jan de 2018.

VUFORIA. 2017. Disponível em: <www.vuforia.com>. Acesso em: 15 dez 2017.