CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA
FACULDADE DE TECNOLOGIA DE CAMPINAS
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

DARA YVE LOPES DA SILVA

DESENVOLVIMENTO DE UM JOGO DE ENTRETENIMENTO COM MOTOR DE JOGOS UNITY UTILIZANDO METODOLOGIAS ÁGEIS

CAMPINAS/SP

2021

CENTRO ESTADUAL DE EDUCAÇÃO TECNOLÓGICA PAULA SOUZA
FACULDADE DE TECNOLOGIA DE CAMPINAS
CURSO SUPERIOR DE TECNOLOGIA EM ANÁLISE E DESENVOLVIMENTO DE SISTEMAS

DARA YVE LOPES DA SILVA

DESENVOLVIMENTO DE UM JOGO DE ENTRETENIMENTO COM MOTOR DE JOGOS UNITY UTILIZANDO METODOLOGIAS ÁGEIS

Trabalho de Graduação apresentado por Dara Yve Lopes da Silva, como pré-requisito para conclusão do Curso Superior de Tecnologia em Análise de Desenvolvimento de Sistemas, da Faculdade de Tecnologia de Campinas, elaborado sob a orientação do Prof. Sandra Aparecida Ribeiro Ossada.

CAMPINAS/SP

2021

RESUMO

Esse trabalho aborda o desenvolvimento de um jogo completo utilizando pontos das metodologias ágeis *Scrum* e *Extreme Programming* que são mais cabíveis ao desenvolvimento de jogos, bem como a integração entre as tecnologias de motor de jogos, softwares para criação artística e a linguagem de programação C#. Trata-se da criação de um jogo completo utilizando das vantagens da integração de tecnologias atuais. Espera-se que o jogo construído seja completo, funcional e testável, e que as soluções propostas pelas metodologias apresentem um impacto positivo no desenvolvimento, tanto na gestão do projeto quanto na parte técnica, trazendo um processo mais organizado e contínuo. *Pixel Frame* é um jogo *pixel art* de plataforma e aventura 2D, feito para computador, no qual o jogador deve chegar ao fim da fase sobrevivendo a uma série de obstáculos e inimigos que encontrará ao longo do caminho.

Palavras-chave: jogo, metodologia ágil, *pixel art*, plataforma, Unity

ABSTRACT

This study addresses the development of a complete game using the agile methods Scrum and Extreme Programming adapted for game development, as well as the integration between game engine technologies, softwares for artistic creation, and the C# programming language. It is about creating a complete game using the advantages of integrating modern technologies. It is expected that the game built will be complete, functional, and testable, and that the solutions proposed by the methodologies have a positive impact on development, both in project management and in the technical part, bringing a more organized and continuous process. Pixel Frame is a pixel art 2D platform and adventure game, made for computers, in which the player must reach the end of the level by surviving a series of objectives and enemies that they will encounter along the way.

Keywords: game, agile methodology, pixel art, platform, Unity

SUMÁRIO

RESUMO 1
ABSTRACT 1
LISTA DE QUADROS 6
1 INTRODUÇÃO 7
1.1 OBJETIVOS GERAIS 8
1.2 OBJETIVOS ESPECÍFICOS 8
1.3 JUSTIFICATIVA 8
2 REVISÃO BIBLIOGRÁFICA 9
2.1 PANDEMIA E JOGOS 9
2.2 DESENVOLVIMENTO DE JOGOS 9
3 MATERIAIS E MÉTODOS 21
3.1 *SOFTWARE ASEPRITE* 21
3.2 *SOFTWARE UNITY* 21
3.3 *VISUAL STUDIO 2019* 22
3.4 *ASSETS* 22
3.5 FERRAMENTA TRELLO 23
4 DESENVOLVIMENTO 24
4.1 REQUISITOS FUNCIONAIS 24
4.2 REQUISITOS NÃO-FUNCIONAIS 24
4.3 DIAGRAMAS 24
4.3.1 Diagrama de caso de uso 24
4.3.2 Diagrama de atividades 25
4.3.3 Diagrama de classes 26
4.4 CONCEITO DO JOGO 26
4.4.1 História 27
4.4.2 Gênero e público-alvo 27
4.4.3 Inspirações 27
4.4.4 Requisitos mínimos de sistema 28
4.5 MECÂNICAS DO JOGO 28
4.5.1 Movimentação 28
4.5.2 Morte 29
4.5.3 Objetos e interações 30

4.6 ELEMENTOS DO JOGO 34
4.6.1 Personagens 34
4.6.2 Fases e níveis 36
4.6.3 Interface 37
4.7 RESULTADOS E DISCUSSÃO 39
4.8 CONSIDERAÇÕES FINAIS 40
REFERÊNCIAS BIBLIOGRÁFICAS 42

LISTA DE FIGURAS

Figura 1 - O ciclo do teste TDD 10
Figura 2 - O processo XP 11
Figura 3 - O processo *Scrum* 13
Figura 4 - Tela inicial do *Super Mario Bros* 16
Figura 5 - Diretório de *assets* com exemplos de *prefabs* (*assets* reutilizáveis) 17
Figura 6 - *Scene* com exemplos de objetos 18
Figura 7 - Exemplo de propriedades de um *GameObject* no painel *Inspector* 18
Figura 8 - Componente *Sprite Renderer* e suas propriedades 19
Figura 9 - Interface gráfica do editor de *Sprite* 19
Figura 10 - Diagrama de caso de uso 25
Figura 11 - Diagrama de atividades 25
Figura 12 - Diagrama de classes 26
Figura 13 - Ristar para *Mega Drive*, 1995 27
Figura 14 - Castlevania por Konami, 1986 28
Figura 15 - Controles para teclado 29
Figura 16 - Componente *Collider2D* 31
Figura 17 - Diferentes tipos de frutas 32
Figura 18 - Armadilha *spike* 32
Figura 19 - Exemplo de uma serra estática 33
Figura 20 - Exemplo de serra móvel 33
Figura 21 - Plataforma de transporte 33
Figura 22 - Bandeira de *checkpoint* 34
Figura 23 - Frame, o personagem principal 35
Figura 24 - Galinha correndo 35
Figura 25 - Cogumelo 35
Figura 26 - Camaleão parado 36
Figura 27 - Camaleão atacando 36
Figura 28 - Primeira fase 37
Figura 29 - Quinta fase 37
Figura 30 - Contadores de vidas e de frutas coletadas 38
Figura 31 - *Cursor* do *mouse* 38

Figura 32 - Interface do menu principal 38
Figura 33 - Interface do menu de pause 39
Figura 34 - Interface do menu de configurações 39

LISTA DE QUADROS

Quadro 1 - Requisitos mínimos de hardware para rodar *Pixel Frame*.....28

1 INTRODUÇÃO

Com a popularização de jogos para computadores e *smartphones*, a comunidade *gamer* aumentou expressivamente, e a cada ano as empresas lançam aparelhos cada vez mais poderosos e com alta capacidade de processamentos gráficos. Alinhada a essa evolução tecnológica, há uma massiva criação de conteúdo em torno da indústria dos jogos, contribuindo para uma densa divulgação de vários títulos. As três principais plataformas de jogos são *mobile*, computador e consoles. A indústria de jogos, no ano de 2020, teve uma receita de aproximadamente U$ 173 bilhões. Desse total, os jogos para dispositivos móveis, por exemplo, levam destaque, contribuindo com quase 57% (BUSINESSWIRE, 2021).

De acordo com análise da empresa *Newzoo* (2021), a taxa de crescimento anual composta do mercado de jogos será de +8,7%, com previsão para chegar a quase US$ 220 bilhões em 2024. A taxa de crescimento anual composta diz respeito ao retorno de investimento durante um determinado período (REIS, 2019). A empresa também diz que o mundo dos jogos terá mais jogadores, e prevê que aproximadamente 2,8 bilhões de pessoas jogarão em um dispositivo móvel.

Além disso, mesmo com o grande impacto global causado pela pandemia da COVID-19, a procura pelo mundo dos jogos recebeu um grande aumento, conforme Clement do portal Statista (2021), justamente por conta do isolamento social e da quarentena. Uma vez que as pessoas passaram a ficar em casa na maior parte do tempo, os jogos se tornaram um dos principais passatempos e meios de entretenimento.

O desenvolvimento e o fluxo de um jogo são diferentes do desenvolvimento de um software convencional. Ao invés de seguir o fluxo de instruções do início ao fim como no convencional, o fluxo do código de um jogo acontece com um *loop* infinito, em que o ponto de parada ocorre quando o usuário decide encerrar o jogo.

A criação de um jogo completo compreende diversas plataformas, áreas e habilidades. Conceitos são consolidados e há implementação de metodologias ágeis durante todo o processo. A criação de jogos também exige comunicação e integração de *frameworks*, pacotes, *assets*, entre outros elementos que devem conversar entre si, conciliando conhecimentos técnico e conceitual. Além disso, há integrações de diversas áreas e tecnologias além da parte de programação, tais como *level design*, *game design*, arte conceitual, produção de áudio, efeitos e dublagens.

O intuito desse projeto é criar um jogo de entretenimento e favorecer aos usuários de jogos mais uma opção de divertimento, aplicando alguns pontos essenciais das abordagens *Scrum* e *Extreme Programming* voltados para projetos de desenvolvimento de jogos. Por serem metodologias voltadas para equipes de desenvolvimento, alguns pontos não serão cabíveis para esse projeto, uma vez que ele será desenvolvido por uma única pessoa.

1.1 OBJETIVOS GERAIS

Este trabalho tem como objetivo o desenvolvimento de um jogo funcional utilizando algumas das ferramentas de desenvolvimento ágil das metodologias *Scrum* e *Extreme Programming*. A criação se dará por meios de tecnologias e áreas de conhecimento diversas, todas integradas para a sua produção.

1.2 OBJETIVOS ESPECÍFICOS

O objetivo é desenvolver um jogo de entretenimento implementando pontos das abordagens do *Game Scrum* e *Extreme Game Development*. Além de integrar diferentes softwares e áreas de estudo com o motor de jogos Unity.

1.3 JUSTIFICATIVA

Com a crescente necessidade da população por entretenimento e a grande popularização dos jogos, resultantes das mudanças que ocorreram de forma abrupta em nosso convívio social por conta da COVID-19, as pessoas passaram a ficar reclusas em suas casas, mantendo o distanciamento social. Buscar alternativas diferentes de entretenimento e socialização pode ser a chave para driblar os efeitos negativos da quarentena. Diante disso, pressupõe-se que o desenvolvimento de um jogo de entretenimento pode colaborar para amenizar os impactos do distanciamento social, trazendo mais uma forma de entretenimento às pessoas.

2 REVISÃO BIBLIOGRÁFICA

2.1 PANDEMIA E JOGOS

Com a chegada avassaladora da COVID-19 no início de 2020, umas das medidas protetivas que a grande maioria da população colocou em prática foram o isolamento e o distanciamento social. A população mundial se encontra reclusa em casa desde o início do ano passado, e esse isolamento tão prolongado sem dúvidas gerou consequências emocionais, psicológicas, e estresse, que impactaram a saúde mental de muitas pessoas.

Este tema se tornou uma das principais preocupações de muitos especialistas da área da saúde e pessoas ao redor do mundo, ao passo que a Organização Mundial da Saúde divulgou um guia para ajudar com os cuidados da saúde mental durante essa pandemia. Manter contato com família e amigos virtualmente é uma forma de aproximação em meio ao distanciamento (OMS, 2020), bem como preservar uma rotina com hábitos saudáveis e momentos de recreação. Uma das recomendações feitas mais especificamente para crianças, mas que também se aplica muito bem a qualquer pessoa, é o incentivo a atividades criativas, passatempos, pois são momentos de descontração e entretenimento.

O mundo dos jogos é um universo que abrange todos esses âmbitos. Com conexão à internet, é possível jogar com amigos e família em uma chamada de voz ou vídeo, pelo computador ou celular. Mesmo sem conexão à internet é possível jogar jogos que funcionem de modo *offline*. Esses foram os principais impulsos por trás da florescente popularização da indústria dos jogos.

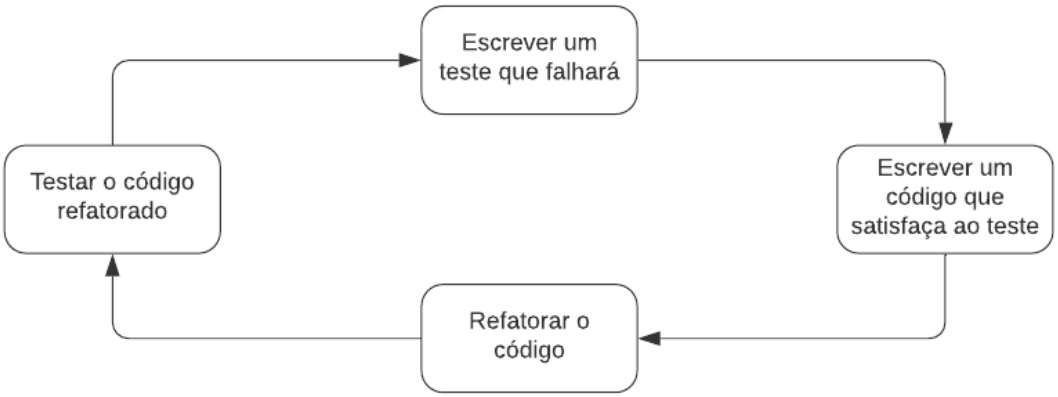
2.2 DESENVOLVIMENTO DE JOGOS

Na área do desenvolvimento, há diversas metodologias e abordagens que se adequam a diferentes tipos de software. Logo, é comum e viável que, dependendo do *software* a ser desenvolvido, uma certa metodologia seja escolhida, ou até mesmo que haja uma mescla de mais de uma metodologia. Na área de desenvolvimento de jogos, duas metodologias são sobressalentes: *Scrum* e *Extreme Programming* (XP). Tais abordagens são utilizadas por compreenderem e atender melhor o processo de desenvolvimento de um jogo, justamente por se tratar de um *software* repleto de integrações e controles de diversos componentes, que devem ser testados e implementados para funcionarem corretamente em conjunto.

Segundo Somerville (2011, p. 44), a metodologia XP traz uma abordagem mais técnica, destacando e enfatizando as boas práticas de programação. Os requisitos recebem o nome de cenários, ou estórias, e são implementados um após o outro, de maneira incremental. Tais tarefas são testadas tanto antes quanto depois de implementadas. Os testes usados são testes TDD (*Test Driven Development*), em que primeiramente são escritos os testes para depois implementar e codificar a funcionalidade, dessa forma há um melhor entendimento sobre o projeto e escrita do código. Dentro do XP, tal prática recebe o nome de desenvolvimento *test-first*.

Conforme Guedes (2019), a ideia por trás do TDD é escrever um teste unitário, antes de começar a programar o código, feito para falhar. Então, deve-se criar um código que satisfaça este teste com êxito. Após essa etapa, uma nova versão deste código deve ser escrita, melhorando e refatorando o primeiro código. Então, o teste é executado mais uma vez, e o novo código deve passar neste teste sem complicações. Tal ciclo está ilustrado na figura 1.

Figura 1 - O ciclo do teste TDD



Fonte: própria autora (2020)

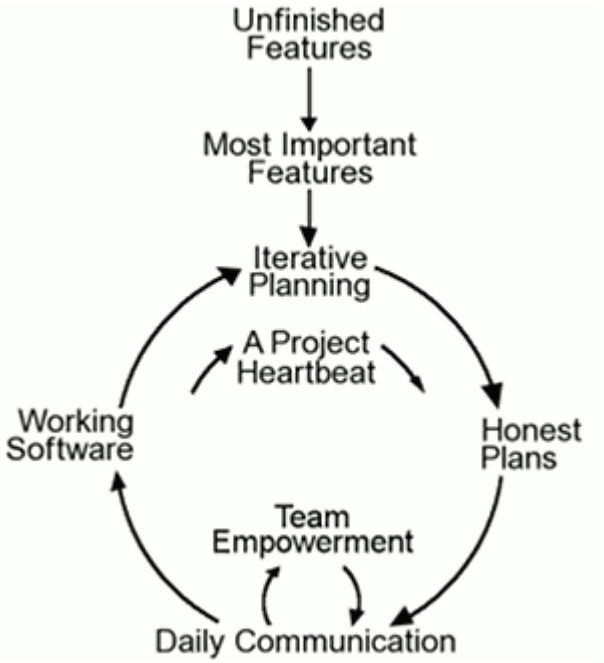
Cada funcionalidade implementada com êxito chama-se *release*. O desenvolvimento incremental se dá com o lançamento e integração desses pequenos *releases*. Sobre a programação, a metodologia sugere que o cliente participe do desenvolvimento e testes de todas as *releases*. Essa participação garante que o desenvolvimento seja mais linear e contínuo. A programação em pares é evidenciada, com a ideia de que os programadores se monitorem e se ajudem a refatorar o código para que haja melhorias sempre.

Segundo Don Wells (2013), a metodologia XP ajuda a entregar um *software* que contém o que é requerido e de forma totalmente funcional, isto é, entrega o que o cliente deseja com qualidade e garantia de que o *software* funciona. Além disso, permite com que o processo de

desenvolvimento responda de maneira positiva a mudanças e a pedidos do cliente, sem que haja maiores riscos, com foco na produção.

A figura 2 ilustra os processos de um projeto utilizando a metodologia XP. O ciclo de revisões é evidenciado, bem como o ciclo de comunicação constante entre todos da equipe. Os requisitos não finalizados são filtrados por importância, e assim que selecionado começa-se o planejamento e desenvolvimento. Na figura 2 também é possível visualizar a constante revisão e aperfeiçoamento justamente por conta dos ciclos de planejamento e principalmente pelo ciclo de comunicação constante.

Figura 2 - O processo XP



Fonte: extremeprogramming.org (2013)

O *Extreme Game Development* (XGD) aparece como uma solução e adaptação do XP voltada para o desenvolvimento de jogos. Essa metodologia foi adaptada para se encaixar em equipes que não são compostas somente por programadores, mas que também contam com diversos tipos de artistas, *game designers*, entre outros. De acordo com Demachy (2003), o *Extreme Game Development* inclui a adição de mídias e conteúdo como uma integração de forma contínua, executando testes automatizados para cada elemento do jogo.

A fim de evitar complicações em um projeto seja por adoção de tecnologia nova, ou mudanças no time de desenvolvimento, o XGD recomenda que todos da equipe compreendam e saibam executar todas as suas tarefas. Por exemplo, em uma empresa de jogos, um certo estilo

de pintura e arte é tido como requisito para o projeto. A equipe de artistas responsáveis deve ser formada por artistas que tenham o mesmo conhecimento nesse estilo. Isso ajuda a evitar maiores prejuízos para o projeto ou para a equipe, pois todos os artistas da equipe são capazes de cobrir o trabalho que falta.

Ao invés de focar numa documentação pesada, há uma grande ênfase em entregar um *software* bem-feito e funcional. Para isso, o time deve trabalhar para apresentar uma versão funcional do jogo o mais rápido possível para que se tenha uma boa ideia sobre as funcionalidades mais essenciais, implementar novas funcionalidades, criar estórias dependendo da necessidade, revisar ferramentas, e incrementar e melhorar o projeto a cada *release*.

Thomas Demachy (2003) comenta que, assim como no XP, o cliente deve estar disposto a trabalhar lado a lado com o time de desenvolvimento, com intuito de não só ajudar a criar e definir as estórias e estimativas de tempo, como também ter certeza de que o projeto está sendo feito de maneira satisfatória e que atende aos requisitos. As estórias são escritas com linguagem simples e descrevem os requisitos e funcionalidades, geralmente são frases rápidas, não muito complexas.

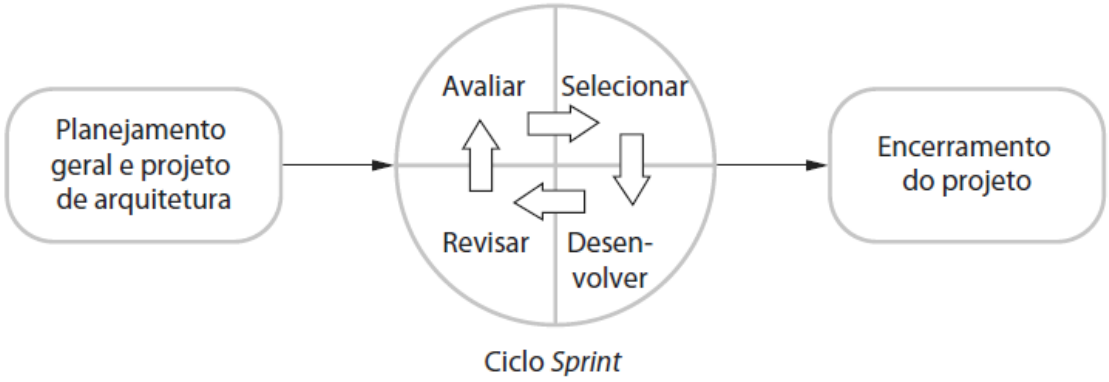
Quanto ao *Scrum*, trata-se de uma abordagem para gerenciamento de projeto, dividida em três fases. De acordo com Sommerville, a primeira fase compreende a parte de definição dos *backlogs*, que são os objetivos, levantamento de requisitos e molde da arquitetura do *software*. O desenvolvimento de um incremento do sistema inicia-se a partir de um *backlog*, feito através de ciclos chamados *sprints*, que geralmente têm de duas a quatro semanas de duração.

Na segunda fase, as ferramentas começam a ser elencadas, e então é iniciada a produção do *software*, com tudo sendo feito através de reuniões diárias com a equipe toda, lideradas pelo *Scrum Master*, a pessoa que lidera o direcionamento do desenvolvimento. Como o *Scrum* não especifica técnicas de programação, fica a critério da equipe escolher a metodologia mais adequada para alcançar os objetivos. Ao finalizar uma *sprint*, a implementação é levada aos *stakeholders*, responsáveis pelo fomento do projeto, para avaliação. A última fase se baseia em montar a devida documentação do projeto e manuais, bem como discutir o aprendizado e avaliar se a execução foi satisfatória.

A figura 3 ilustra o ciclo de *sprints* com as etapas de desenvolvimento de cada tarefa. A primeira parte abrange o planejamento do projeto, com definições de requisitos, ferramentas, serviços, conversas com cliente e time de desenvolvimento. Para cada *sprint* iniciada, os mesmos processos são executados. A última etapa diz respeito ao fechamento do projeto, com

atualização da documentação, reuniões para avaliar o andamento do projeto, recolher opiniões da equipe e analisar rendimento e o que pode ser melhorado para aplicar ao próximo projeto.

Figura 3 - O processo *Scrum*



Fonte: Sommerville (2011)

No âmbito do desenvolvimento de jogos, o ideal é encontrar uma abordagem – ou um conjunto de abordagens – que minimize erros e riscos durante o desenvolvimento, concedendo flexibilidade para reagir às mudanças de projeto. Schofield (2007) escreve que tais mudanças podem ocorrer devido a diversos fatores, tais como editores, administração e gestão e até mesmo o próprio time de desenvolvimento. De acordo com Godoy e Barbosa (2010), uma boa mescla entre XGD e *Scrum* se mostra muito adequada ao desenvolvimento de jogos. Juntas, ambas as abordagens contribuem para o processo de descobrir e esculpir o "fator diversão" de um jogo.

Com intuito de moldar melhor o fator diversão e ajudar a distribuir melhor as tarefas durante os ciclos de cada implementação, o *Game Scrum* aparece como uma solução. Godoy e Barbosa (2010) apresentam as três fases do *Game Scrum*: pré-produção, produção e pós-produção, que seguem uma estrutura similar às três fases do *Scrum*. A pré-produção diz respeito à fase de planejamento geral. É neste momento que se estipulam quais os objetivos do jogo, monta-se a história a ser contada – se houver, discute-se aparência e identidade visual a serem alcançadas, e determina-se o fator diversão.

É feito um *brainstorm* para coletar todas as ideias. É coberta a parte da identidade visual do jogo: *game design*, interface de usuário, artes conceituais, ou seja, a aparência geral do jogo. Esboços de personagens, cenários, jogabilidade, itens, estilo de jogo são criados. Também se discute funcionalidades gerais e começa-se a desenvolver o primeiro protótipo, com

características simples e principais funcionalidades. Assim que usado, geralmente este protótipo é descartado. Todos os elementos apresentados acima são compilados em um documento de *design* do jogo, que ajuda a guiar o escopo do projeto, evitando atrasos e grandes riscos. De acordo com os autores, a produção deste documento é opcional para projetos menores. É válido lembrar que este documento é suscetível a mudanças, e deve ser atualizado conforme as reuniões e adaptações feitas até a última fase de desenvolvimento.

O estágio de produção trabalha em cima de um escopo já bem definido do projeto. Godoy e Barbosa (2010) descrevem que o documento de design do jogo se transforma no *product backlog*. Diferentemente do *Scrum*, em que todo o trabalho deve ser finalizado ao fim da *sprint*, o *Game Scrum* flexibiliza as iterações, de modo que pode haver trabalho a ser concluído mesmo após o término de um ciclo, como é o caso das criações artísticas. Os dois autores também recomendam mesclar *Scrum* e XP para o desenvolvimento, adaptando e escolhendo a abordagem que melhor compreender as necessidades definidas no planejamento geral.

A fase de pós-produção, o jogo é testado para avaliar o fator diversão e o funcionamento do *software* como um todo. Assim como no *Scrum* convencional, nessa fase são colhidas avaliações e opiniões sobre tudo que foi aprendido durante o processo e sobre o que pode melhorar, bem como sugestões para melhorar em projetos futuros, análise das ferramentas utilizadas e conversar sobre os problemas e dificuldades encontrados.

O fator diversão é um termo usado para descrever a quantidade de diversão que um jogo traz. Encontrar o fator diversão é uma tarefa desafiadora, uma vez que não há uma fórmula nem um algoritmo que mostre qual o fator diversão de um jogo; descobre-se jogando (HERN, 2002). A ideia trazida pelo XGD de desenvolver o mais rápido possível um protótipo se baseia justamente na procura por esculpir e melhorar o fator diversão, uma versão testável das funcionalidades básicas do jogo ajuda no processo de melhoria dos elementos do jogo a cada iteração.

Hern (2002) destaca a importância de um bom fator diversão: é o fator mais importante e central de um jogo. Sem ele, o jogo está fadado a falhar – ou, num melhor caso, atingir sucesso aleatoriamente. O foco nesse fator é imprescindível, pois é uma tarefa muito delicada e que facilmente pode encontrar complicações. A implementação de uma possível solução ao problema de melhorar o fator diversão pode atrapalhar ou bloquear outro aspecto do jogo, isto é, há chances de surgirem incompatibilidades a cada implementação.

O autor ainda discute que uma série de fatores que auxiliam a moldar uma experiência divertida do usuário. Ter criatividade é o primeiro passo para elencar soluções. Analisar cada

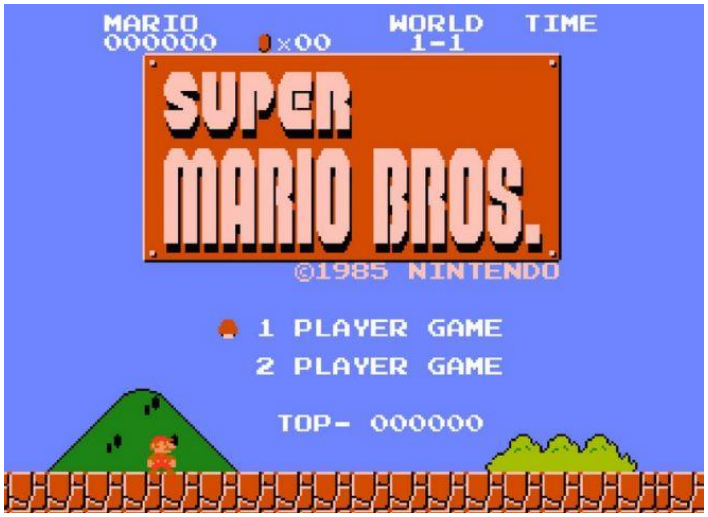
caso e pensar numa solução adequada é essencial. Quando se trabalha em equipes, o ponto de vista de cada um conta muito para o processo. Outro ponto é deixar claro para o jogador o que ele deve fazer, para que ele não fique perdido ao jogar. A menos que isso faça parte da premissa e conceito do jogo, é crucial ter guias, tutoriais, instruções mostradas através da interface. E mesmo em um caso como este, em que a premissa gira em torno de deixar que o jogador explore sem um rumo definido, mostrar instruções sobre controles ou explicação da interface de usuário são indispensáveis. Se o jogador não faz ideia do que deve fazer, o fator diversão fica completamente comprometido (HERN, 2002).

Quanto mais interação um jogo propõe, mais divertido se torna. Hern aborda essa ideia sugerindo que o jogo apresente opções que, de certa forma, dão poder para o usuário alterar ou customizar elementos dentro do jogo. Entretanto, as regras do jogo são o que define o quanto o jogador pode interagir ou modificar partes dele, o que é importante, já que um jogo completamente destrutível ou modificável pode perder o interesse rapidamente com o tempo.

Outro ponto apresentado é o fator desafio. Um jogo se torna divertido quando os desafios são difíceis, porém possíveis de serem completados. Nesses casos, diversas habilidades do jogador são desafiadas a melhorar a cada desafio, para que ele consiga passar para o próximo nível. O fator desafio está ligado de certa forma com o ego do jogador, uma vez que os desafios podem testar diversas habilidades suas, tais como: estratégia, habilidade motora, audição, e às vezes até mesmo paciência e calma (HERN, 2002).

Quando um jogo trabalha as emoções do jogador, seja por instaurar tensão, apresentar uma história comovente ou causar sustos e medo, o fator diversão aumenta bastante, podendo fazer o jogador criar laços com o jogo – o que pode ser um dos objetivos caso a empresa pretenda lançar uma série de jogos. Ilustrado na figura 4 está um ótimo exemplo de uma franquia que tem um poderosíssimo fator diversão é a série de jogos *Super Mario Bros*. Segundo o site oficial da Nintendo, criadora do *Super Mario Bros*, o primeiro jogo da série foi lançado em 1985.

Figura 4 - Tela inicial do *Super Mario Bros*



Fonte: Nintendo (1985)

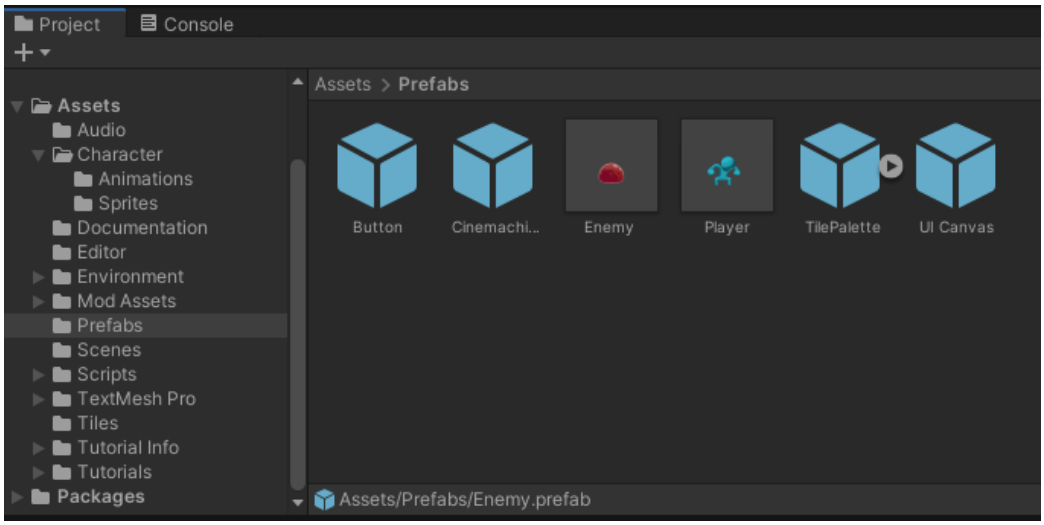
Sem dúvidas o *Super Mario Bros* é uma das séries mais bem-sucedidas e poderosas do mundo do videogame, mesmo após 35 anos de lançamento. Isso se deve ao excelente trabalho de toda a equipe desenvolvedora ao esculpir o fator diversão, montar os *designs* de personagem, cenário, música, jogabilidade, dar ao jogador diversas possibilidades de interação com os elementos do jogo e capacidade de exploração (através de níveis e fases ocultos desbloqueáveis, por exemplo).

Outro fator que contribui para o sucesso de um jogo é fazer com que o jogador se identifique de certa forma com algum elemento do jogo, como seu próprio avatar, por exemplo. Hern (2002) comenta que dar motivações, personalidade e sentimentos aos personagens, histórias e escolhas, fazem com que o jogador queira cada vez mais jogar. Não há um jeito correto de escolher elementos para construir a diversão de um jogo, por isso diversos ciclos de comunicação e testes são essenciais ao longo do desenvolvimento. Por isso a importância de usar metodologias e abordagens que permitam adaptações às mudanças que ocorrerão durante o projeto.

Jogos são desenvolvidos com *softwares* especializados chamados motores de jogos, ou *game engines*, que contêm um conjunto gigante de ferramentas essenciais para o desenvolvimento de jogos. Segundo Marques (2020), é opcional utilizar ou não uma *engine* para a produção de um jogo. Dentre os diversos *softwares* disponíveis, há o motor de jogo chamado Unity. Muito conhecido e utilizado na indústria de jogos, é possível trabalhar tanto com projetos 2D quanto 3D com essa ferramenta. A principal linguagem de programação usada no motor Unity para programação de *scripts* é C#.

De acordo com o manual do Unity (2020b), um cenário é composto por vários itens, chamados *assets*. Um *asset* pode ser um arquivo de áudio, uma imagem ou um modelo 3D (UNITY, 2020a), por exemplo. É importando esses itens que se constrói todo o ambiente do jogo. É possível também adquirir *assets* direto da loja de *assets* do Unity. A figura 5 exemplifica o painel que contém os *assets* de um projeto.

Figura 5 – Diretório de *assets* com exemplos de *prefabs* (*assets* reutilizáveis)

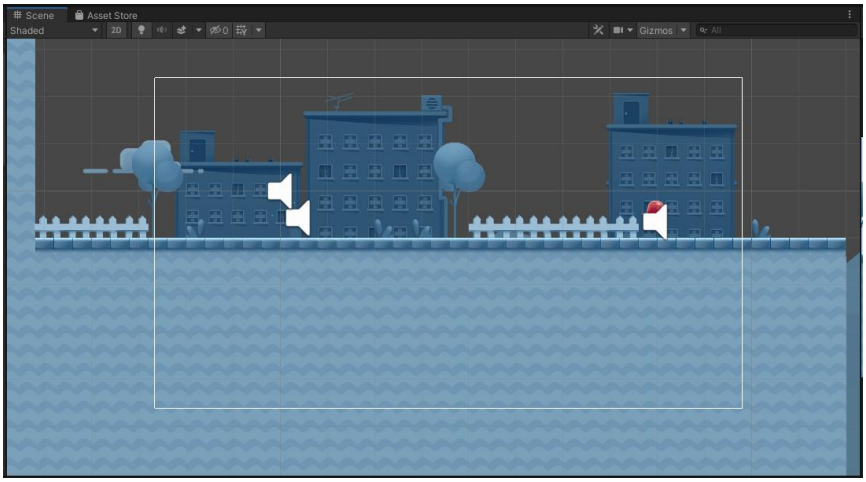


Fonte: própria autora (2020)

As *Scenes* são as cenas, os ambientes de um jogo. Um jogo simples, por exemplo, pode ser construído em apenas uma *scene*, diferentemente de um jogo maior e mais complexo, em que se deve separar cenários e fases do jogo em várias cenas diferentes. Cada cena contém decorações, ambientes, luz, entre outros elementos que formam o cenário do jogo (UNITY, 2020b). Tal processo de construção dos cenários e cenas é chamado de *level design*. Geralmente uma empresa de desenvolvimento de jogos com equipes de desenvolvedores conta com artistas especializados em decorar e construir os ambientes do jogo.

Conforme o manual (2020b), quando um *asset* é inserido na cena, ele recebe propriedades completamente customizáveis, tornando-se objetos chamados *GameObjects*. É através do painel *Inspector* que se atribui propriedades, componentes, e se transforma o objeto em uma personagem, ambiente ou efeito especial. Além de aceitar importar *assets* criados e moldados por outros softwares de criação e modelagem, Unity também conta com uma série de *assets* pré-construídos e *scripts* pré-programados para agilizar algumas tarefas e poupar trabalho. A figura 6 mostra a área reservada para uma *scene* e seus objetos.

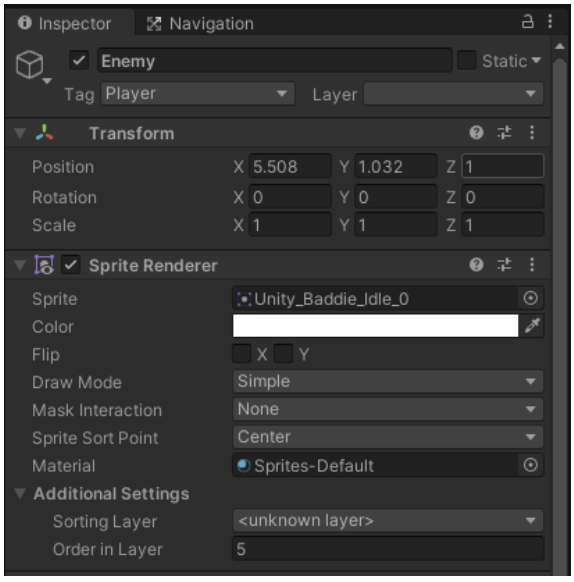
Figura 6 - *Scene* com exemplos de objetos



Fonte: própria autora (2020)

Como visto, uma cena é composta por *GameObjects*. Tudo sobre o objeto pode ser customizável: pode-se aplicar propriedades que lidam com física, rotação, colisão, tamanho, pode-se atribuir um arquivo de áudio ao objeto, ou então arquivos de scripts. Com um script, os objetos passam a ter funcionalidades e ações, porém não são todos que precisam, como por exemplo objetos decorativos ou paisagens. A figura 7 ilustra o painel contendo as propriedades customizáveis aplicadas a um *GameObject*.

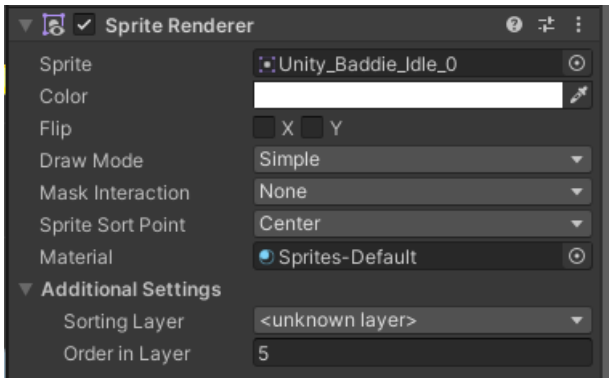
Figura 7 – Exemplo de propriedades de um *GameObject* no painel *Inspector*



Fonte: própria autora (2020)

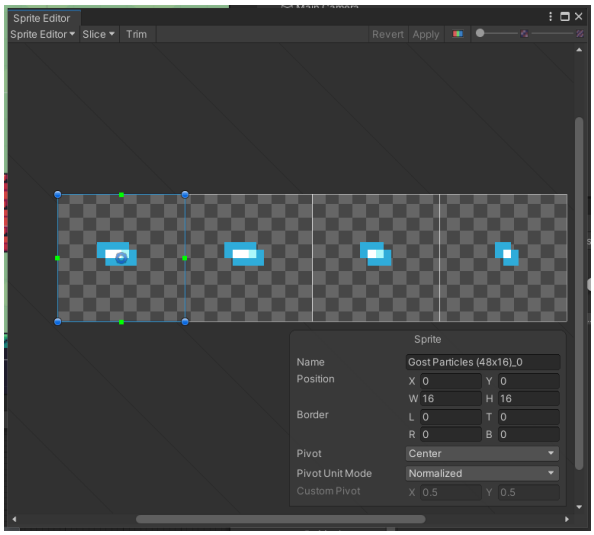
Durante a criação de jogos 2D, utiliza-se objetos gráficos chamados *Sprites*, imagens bidimensionais sobrepostas usadas para representar as personagens (JACOB, 2020). A partir dessas *sprites* é possível fazer a animação dessas personagens. Na maioria das vezes, é mais vantajoso e conveniente colocar todas as *sprites* em um arquivo único de imagem. Esse arquivo recebe o nome de *sprite sheet*. A partir de uma *sprite sheet*, é possível então recortar e extrair as *sprites* para utilizá-las. Dentro do Unity há uma ferramenta específica para esta tarefa chamada *Sprite Editor*, como mostra a figura 9. Ilustrado na figura 8 está o componente *Sprite Renderer*, através do qual as *sprites* são renderizadas na tela, que deve ser ligado ao *GameObject* segundo o manual de jogabilidade 2D da Unity (2020c).

Figura 8 - Componente *Sprite Renderer* e suas propriedades



Fonte: própria autora (2020)

Figura 9 – Interface gráfica do editor de *Sprite*



Fonte: própria autora (2021)

De acordo com o site da própria Unity (2020e), mais de 50% dos jogos foram desenvolvidos com o motor Unity no ano de 2019 – somando os jogos para PC, console e dispositivos móveis – e mais de 120 milhões de jogadores em média por mês utilizaram os serviços de texto e voz da Unity. Além disso, no mesmo ano, foi calculado que mais de 20 plataformas diferentes suportavam as criações feitas com o Unity.

O motor de busca se destaca por ser uma ferramenta completa e gratuita. É amplamente usado pelas mais diversas empresas no ramo dos jogos por conta de sua versatilidade e robustez, e recebe constantes atualizações, correções e implementações. Conhecer as principais ferramentas deste *software* possibilita desenvolver qualquer tipo de jogo, seja ele 2D ou 3D.

3 MATERIAIS E MÉTODOS

Para o desenvolvimento do jogo, foi utilizado o *software* Aseprite para a criação de elementos, itens, personagens, animações e demais *assets*. Esse *software* é uma ferramenta criada especialmente para artes feitas com *pixels*, denominadas *pixel art*, o que facilita o processo de criação de artes do jogo. O Unity foi o motor de criação do jogo, que traz uma interface amigável e conta com uma loja de elementos e recursos de muita qualidade para serem utilizados; tais recursos são disponibilizados pelos usuários da comunidade Unity. A programação dos *scripts* foi feita na linguagem C#, tida como padrão para os *scripts* utilizados no motor Unity. O ambiente de desenvolvimento integrado (IDE) utilizado foi o *Visual Studio* 2019, por conta de alguns recursos que ajudam na produtividade.

Por ser um jogo com escopo controlado e requisitos bem definidos, foi possível que a própria autora atuasse nas áreas de programação e criação de artes. O *software* Unity conta com suporte de arquivos de som e animações, sendo assim a ferramenta principal neste projeto. Como mencionado, a loja de recursos do Unity conta com pacotes de artes e animações gratuitas que foram utilizadas para agilizar o desenvolvimento, o que permitiu ter um foco na programação e na montagem das *scenes*, modelos e *prefabs*.

3.1 *SOFTWARE* ASEPRITE

Para criar e editar as *pixel arts* do jogo, o programa Aseprite foi utilizado por ser uma ferramenta desenvolvida especialmente para *pixel art*. Considerada por muitos como a melhor ferramenta para artes com *pixel*, ele facilita no processo de criação e manipulação de *sprites* e animações, por exemplo, com ferramentas e atalhos que tornam o trabalho mais ágil.

A comunidade de artes e artistas *pixel art* é bastante vasta, portanto, há uma variedade de conteúdos e artes gratuitos de qualidade disponíveis, que cobrem diversos assuntos sobre este estilo de arte. No Aseprite, é possível importar arquivos de imagem, por exemplo, e fazer customizações próprias de animações e *sprites*. Depois, as opções de exportação das artes feitas foram importadas e manipuladas dentro do motor Unity.

3.2 *SOFTWARE* UNITY

Como abordado anteriormente, o motor de jogos Unity é uma poderosa ferramenta gratuita e cabível para o desenvolvimento de um jogo plataforma simples. Uma das suas grandes vantagens é sua grande e ativa comunidade. A *Unity Asset Store* é uma biblioteca *online* em que é possível obter *assets* gratuitos e comerciais criados pela Unity e por diversos contribuidores da comunidade.

Essa loja cobre uma variedade muito grande de *assets*, que vão desde animações, modelos, artes e texturas até sistemas de salvamento de jogo. Tudo isso contribui para um desenvolvimento mais rápido, poupando uma carga de trabalho e tempo. Tais *assets* adquiridos podem ser usados a partir do *Package Manager* dentro do Unity.

3.3 *VISUAL STUDIO 2019*

O *Visual Studio* foi o ambiente de desenvolvimento utilizado para escrever os *scripts* do jogo, pois tem suporte completo às bibliotecas do Unity. Junto a isso, seu recurso mais útil é o *IntelliSense*, um termo usado para diversos recursos para edição de código, tais como autocompletar, informações sobre parâmetros e listas, entre outros. O *IntelliSense* aumenta bastante a produtividade e a precisão do código. Uma das linguagens às quais o *Visual Studio* é dedicado é C#, a linguagem usada neste projeto.

3.4 *ASSETS*

Os *assets* escolhidos foram recolhidos tanto da loja de *assets* da comunidade Unity quanto do site *itch.io*, voltado para jogos e pacotes para desenvolvimento de jogos. *Pixel Adventure* 1 e 2, *Treasure Hunters* e *Kings and Pigs* são pacotes *pixel arts* de *sprites* do artista *Pixel Frog*, que estão respectivamente disponibilizados na loja do Unity, e no site *itch.io*. Como são *assets* gratuitos com licença livre, algumas artes foram modificadas para melhor se adequar ao tema de *Pixel Frame*. A fonte usada no jogo vem do pacote *Treasure Hunters*. São pacotes muito completos, que possuem desde elementos gráficos para construir interfaces até *sprites* com animações prontas para uso. O *cursor* do *mouse* pertence ao pacote *MiniFX* do artista *GrafxKid*, disponível gratuitamente no site *itch.io*.

Os efeitos sonoros são do pacote *Casual Game SFX Pack*, de *Dustyroom*, e a música de fundo é do pacote de músicas *8Bit Music Album*, de *GWriterStudio*, ambos disponíveis gratuitamente na *Unity Asset Store*.

3.5 FERRAMENTA TRELLO

Trello é uma ferramenta para gerenciamento de projeto, em que listas e cartões podem ser criados para organizar as tarefas e definir prioridades. Para este projeto, quatro listas foram criadas: *backlog*, tarefas em andamento, tarefas concluídas, e seção de *links* e arquivos referentes à documentação do projeto. Cada cartão recebeu etiquetas para ajudar na priorização de tarefas e organização das partes do projeto.

4 DESENVOLVIMENTO

4.1 REQUISITOS FUNCIONAIS

A seguir serão descritos os requisitos funcionais do jogo:

a. O jogador pode dar pulos duplos (*double jump*);
b. O jogador possui três vidas;
c. Os inimigos são eliminados quando o jogador pula em cima deles;
d. Ao ser atingido, o jogador perde uma vida;
e. Salvar o progresso de jogo e as opções selecionadas nos menus;
f. Permitir que o jogador altere a resolução do jogo;
g. Ter um sistema de pontuação.

4.2 REQUISITOS NÃO-FUNCIONAIS

A seguir serão descritos os requisitos não-funcionais do jogo:

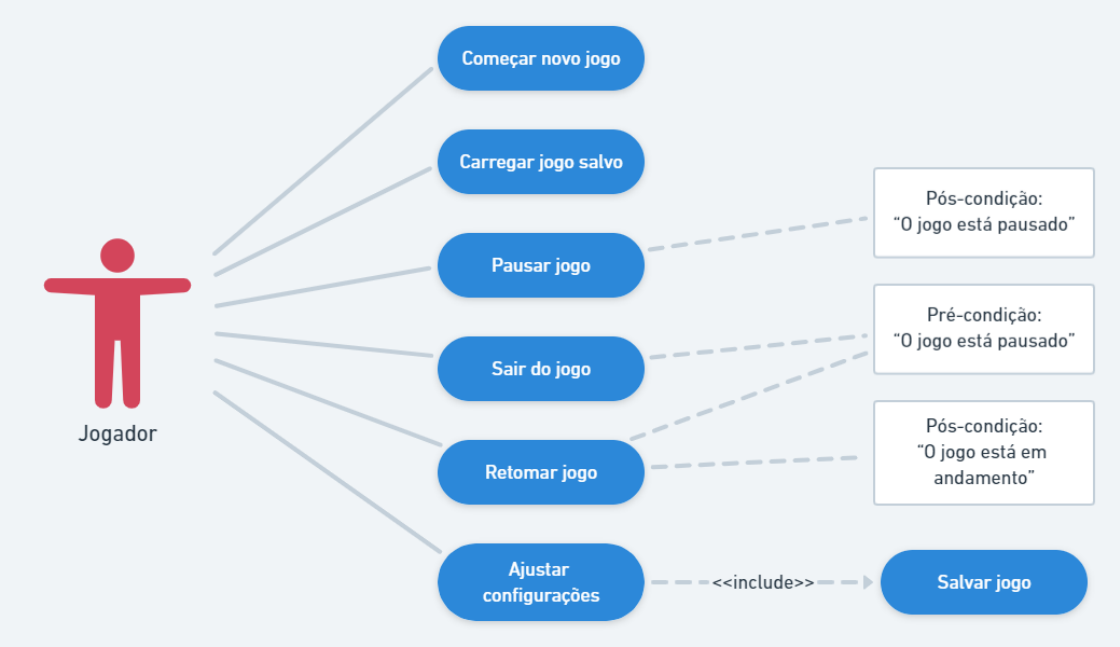
a. O jogo deve rodar em computadores (*Windows*, *Linux*, *MacOS*);
b. As resoluções do jogo devem ser compatíveis com as resoluções disponíveis à tela do usuário;
c. Para salvar progressos do jogo, deve-se utilizar a classe *PlayerPrefs* do Unity;
d. O progresso das fases deve ser salvo a cada *checkpoint* atingido;
e. As configurações devem ser salvas assim que alteradas;
f. Caso o jogador perca suas três vidas, ele deve ser colocado no começo da fase;
g. Programação deve ser feita em C#;
h. Não deve haver limite de pontuação para cada fase.

4.3 DIAGRAMAS

As figuras 10, 11 e 12 ilustram, respectivamente, os seguintes diagramas: caso de uso, atividades e classes.

4.3.1 Diagrama de caso de uso

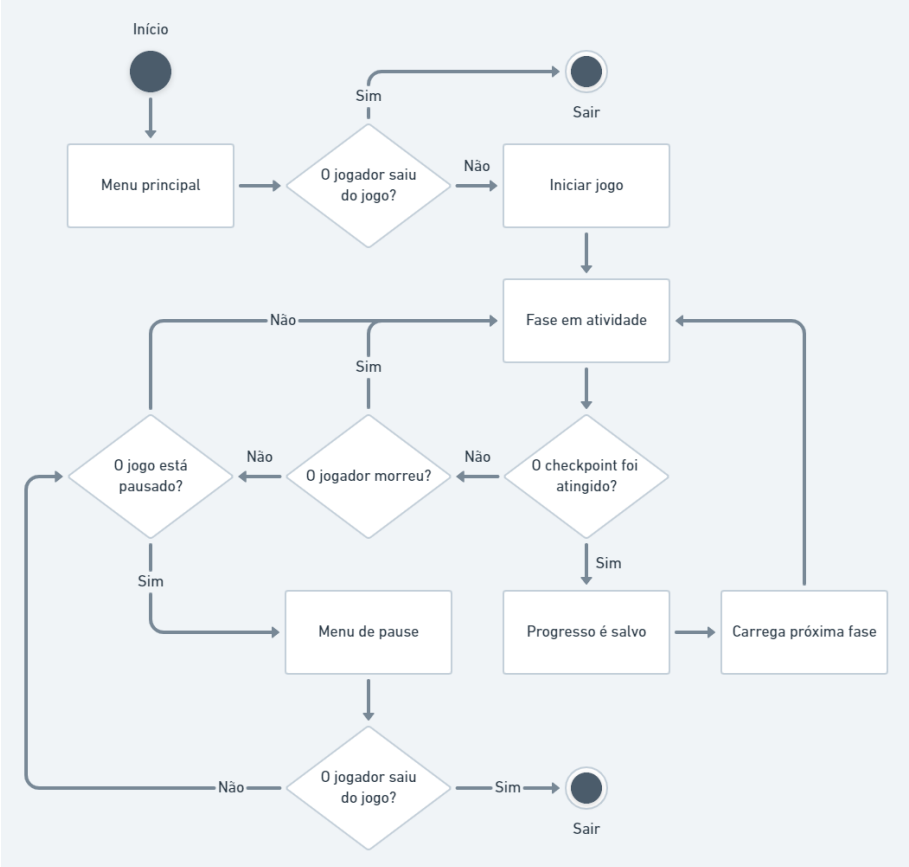
Figura 10 - Diagrama de caso de uso



Fonte: própria autora (2021)

4.3.2 Diagrama de atividades

Figura 11 - Diagrama de atividades

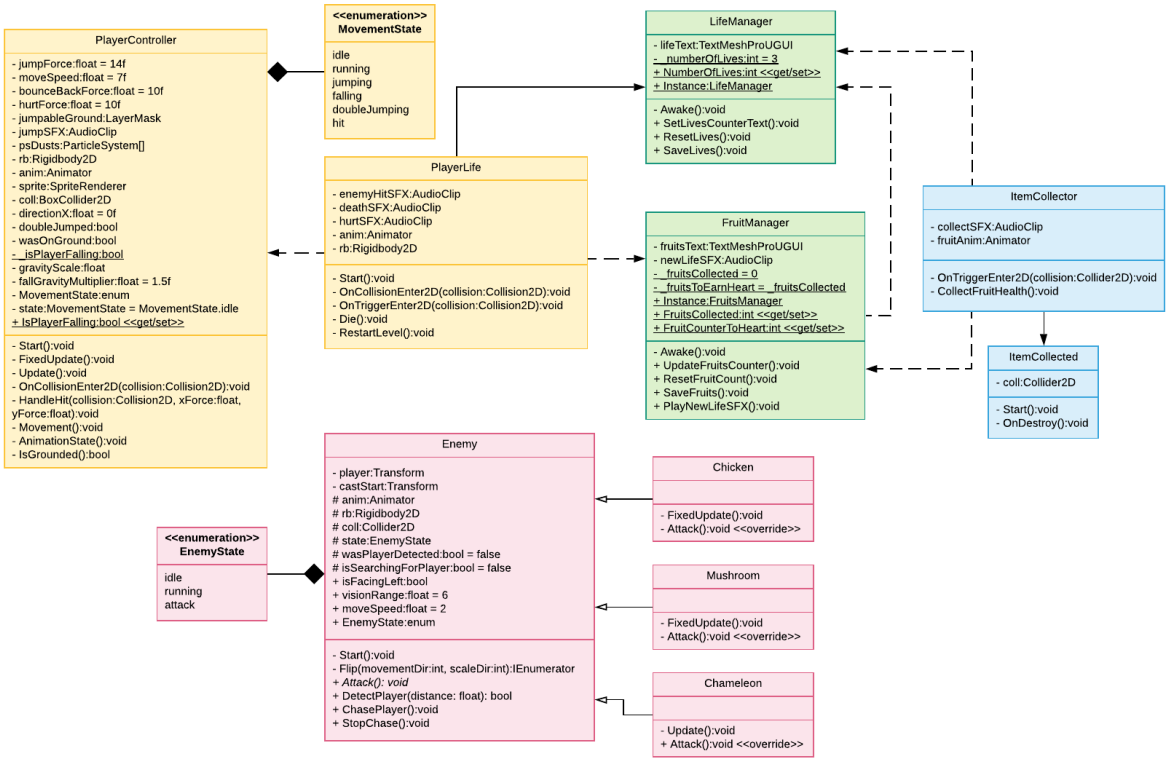


Fonte: própria autora (2021)

4.3.3 Diagrama de classes

Todos os *scripts* criados no Unity são herdeiros da classe base *MonoBehaviour*, que oferece algumas funções de ciclo de vida da aplicação que facilitam durante o desenvolvimento do jogo. Uma vez que todas as classes ilustradas na figura 12 herdam de *MonoBehaviour*, retratar todas essas relações deixaria o diagrama poluído, portanto foram omitidas. Estão mapeadas na figura 12 as principais classes relacionadas à movimentação e vida do jogador, às interações, e aos inimigos.

Figura 12 - Diagrama de classes



Fonte: própria autora (2021)

4.4 CONCEITO DO JOGO

Pixel Frame é um jogo plataforma em que o jogador deve chegar ao fim de cada fase coletando frutas especiais e desviando de obstáculos e inimigos que constantemente aparecerão ao longo do caminho. Ao coletar as frutas, o jogador ganha uma vida bônus. O gênero do jogo é plataforma de ação e aventura, em que o jogador deve chegar a um *checkpoint* atravessando um mapa com terreno repleto de armadilhas, empecilhos, e inimigos, os quais devem ser

evitados e/ou eliminados. O jogador deve coletar o máximo de frutas que conseguir durante o percurso.

4.4.1 História

Frame, o personagem principal, precisa coletar e comer frutas para manter-se vivo. Durante o caminho, inimigos e obstáculos aparecerão para tentar impedi-lo e dificultar sua jornada.

4.4.2 Gênero e público-alvo

É um jogo de plataforma e aventura, cujo público-alvo são pessoas que gostem de tais gêneros de jogo, bem como o estilo de arte *pixel art*.

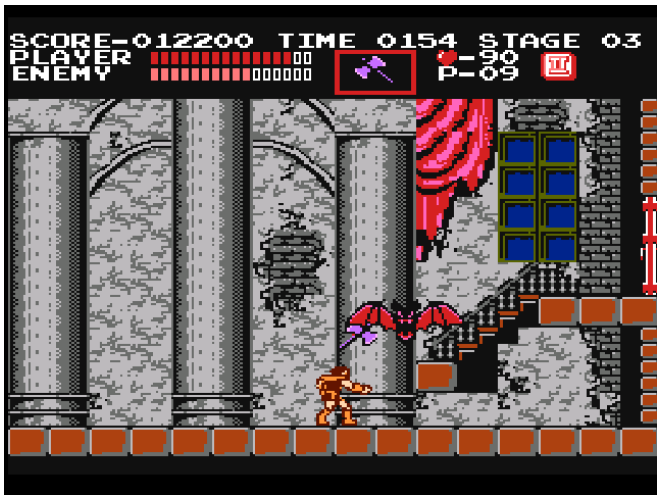
4.4.3 Inspirações

O jogo tem inspirações em jogos antigos de plataforma, como o *Super Mario Bros*, *Castlevania* e *Ristar*, por exemplo. Estes dois últimos estão demonstrados nas figuras 13 e 14. São jogos em que o ponto principal é fazer com que o jogador atravesse terrenos de níveis e fases, derrotando inimigos e passando por obstáculos. As influências para o estilo *pixel art* certamente vêm dos jogos de gerações passadas, em que umas das únicas opções – senão a única – era construir as interfaces e artes com *pixels*, por conta da área pequena das telas e paleta de cores mais limitadas (JACOB, 2020).

Figura 13 - Ristar para *Mega Drive*, 1995

Fonte: Steam (2010)

Figura 14 - Castlevania por Konami, 1986



Fonte: Daniel Gularte (2010)

4.4.4 Requisitos mínimos de sistema

Atualmente, o jogo está projetado apenas para computadores. Os requisitos mínimos de sistema para rodar o jogo estão descritos no quadro 1, conforme a documentação da Unity.

Quadro 1 - Requisitos mínimos de hardware para rodar Pixel Frame

Sistema operacional	Versão do sistema operacional	CPU	API de gráficos
Windows	Windows 7, Windows 10 e Windows 11	Arquitetura x86, x64 com suporte ao conjunto de instruções SSE2	Compatível com DX10, DX11 e DX12
Linux	Ubuntu 20.04, Ubuntu 18.04 e CentOS 7	Arquitetura x64 com suporte ao conjunto de instruções SSE2	OpenGL 3.2+, compatível com Vulkan
Mac OS	High Sierra 10.13+	Arquitetura x64 com SSE2	GPUs Intel e AMD com compatível com Metal

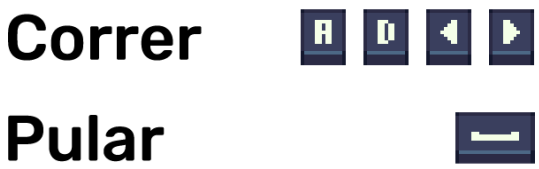
Fonte: Unity Documentation (2021d)

4.5 MECÂNICAS DO JOGO

4.5.1 Movimentação

Os controles do personagem são simples e intuitivos: para mover-se para a esquerda e direita, utiliza-se as teclas A e D, ou as setas do teclado. Para pular, a tecla espaço é usada. O jogador pode apertar o botão de pular duas vezes seguidas para fazer um pulo duplo. A altura do pulo varia conforme o tempo que a tecla fica pressionada. Se o jogador pressionar por pouco tempo, o pulo é mais curto; se pressionar por mais tempo, o pulo é mais longo. Também é possível mover-se para os lados enquanto o jogador está no ar. A figura 15 ilustra os controles para teclado.

Figura 15 - Controles para teclado



Fonte: própria autora (2021)

4.5.2 Morte

O jogador inicia o jogo com três vidas, que não se restauram automaticamente quando um *checkpoint* é atingido. Ao ser atingido por um inimigo ou chocar-se contra um obstáculo, uma vida é perdida. O jogador morre quando perde todas as suas vidas, sendo obrigado a recomeçar a fase.

4.5.2.1 Sistema de *save*

Para salvar parâmetros de preferências do jogador e os progressos de jogo, foi utilizada a classe *PlayerPrefs*, oferecida pelo Unity, que armazena chaves nomeadas e seus valores. Tais chaves são salvas no registro do sistema operacional do computador. Por conta da facilidade em acessar o registro e manipular o conteúdo das chaves, *PlayerPrefs* não é o método mais indicado de salvar informações importantes de progresso do jogo como pontuação, *checkpoints* e número de vidas. É mais recomendado para salvar configurações gerais como volume e resolução. Entretanto, como *Pixel Frame* está nas fases iniciais de desenvolvimento e prototipação, por ora essa ferramenta é uma ótima opção.

A classe armazena apenas três tipos de dados: inteiros, *floats* e *strings*. Alguns parâmetros precisaram ser convertidos por causa dessa limitação. Os botões *toggle*, usados para alternar entre os estados "ativo" e "não-ativo", são os componentes usados no jogo para ativar e desativar a música e/ou os efeitos sonoros. Uma das propriedades desse componente chama-se "*Is On*", a qual armazena esses estados usando um valor do tipo *boolean*; ou seja, verdadeiro ou falso.

Para armazená-lo com *PlayerPrefs*, foi feita uma conversão para inteiros: caso "*Is* O*n*" seja verdadeiro, a informação é salva como "1"; caso seja falso, como "0". A mesma lógica é seguida para carregar essas informações e convertê-las para *boolean* novamente. Os dados alterados no menu de configurações são salvos assim que modificados, sem a existência de um botão específico para salvar.

Na classe *SettingsMenu*, criada para gerenciar o menu de configurações, foi criado um vetor que armazena todas as resoluções colhidas disponíveis para o jogador escolher. A partir desse vetor, é possível acessar cada resolução através de seu índice, um número inteiro que representa a posição de um determinado dado no vetor. Este índice é salvo no *PlayerPrefs*, e, para carregar a resolução selecionada, uma vez que se tem a posição dela no vetor, basta resgatá-la a partir desse índice.

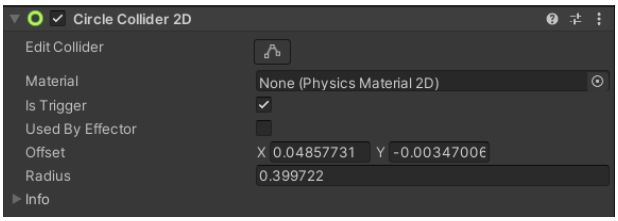
Quando o jogador atinge o *checkpoint* de uma fase, a próxima fase (*scene*) é carregada e salva, bem como o número de vidas que o jogador possui. No Unity, cada *scene* também possui um índice, o qual é salvo e carregado quando o jogador seleciona a opção "Continuar jogo" no menu principal. Ao clicar em "Novo jogo", a primeira fase é sempre carregada. Por fim, a pontuação é gerenciada como um inteiro, sendo carregada quando o jogador escolhe "continuar jogo".

4.5.3 Objetos e interações

Para permitir interações com o sistema de física 2D, pode-se conectar um componente chamado *Collider2D* a um objeto, apresentado na figura 16. Há diversos tipos de *colliders*: *Box Collider2D*, *Circle Collider2D*, *Polygon Collider2D*, entre outros. Cada um tem um formato diferente para se ajustar melhor ao formato do objeto ao qual está conectado. Uma de suas propriedades é fazer com que o objeto seja um *trigger* (acionador). Um *trigger* permite que outros objetos o atravessem, desencadeando uma determinada ação programável. Um objeto sem essa propriedade ativa apresenta característica de um colisor (*collider)*, ou seja,

literalmente uma colisão ocorrerá entre os objetos, um não atravessará o outro. Também é possível programar uma ação quando essa colisão acontece. Os dois métodos que permitem escrever comandos quando essas interações acontecem são *OnTriggerEnter2D* e *OnCollisionEnter2D*.

Figura 16 - Componente *Collider2D*



Fonte: própria autora (2021)

Atualmente *Pixel Frame* conta com objetos que são coletáveis ou armadilhas. Os coletáveis são *triggers*, pois são objetos que serão atravessados pelo jogador, sem acontecer uma colisão, como as frutas, por exemplo. Quando essa interação acontece, a pontuação aumenta, e a fruta coletada é destruída da *scene*. Já as armadilhas são *colliders*, que eliminam vidas do jogador quando uma colisão entre eles ocorre. Note que é possível ter vários tipos de armadilhas com diferentes efeitos. Para diferenciá-los é possível adicionar uma *tag* a um objeto, uma etiqueta nomeada que facilita a filtragem de diferentes tipos de objetos. As *sprites* das armadilhas e do *checkpoint* são do pacote *Pixel Adventure 1*.

4.5.3.1 Frutas

As frutas aparecem durante todo o percurso de uma fase. Para coletá-las, basta que o jogador passe por cima delas. Cada fruta equivale a um ponto. Quando cinco frutas são coletadas, uma nova vida é fornecida ao jogador. A melancia concede uma vida instantânea ao jogador, mas não aparece com frequência. Na figura 17, é possível ver as quatro diferentes frutas: laranja, kiwi, morango e melancia. As *sprites* são do pacote *Pixel Adventure 1*.

Figura 17 - Diferentes tipos de frutas

Fonte: própria autora (2021)

4.5.3.2 *Spikes*

Como ilustra a figura 18, *spikes* são um tipo de armadilha que são pontiagudas e fixas no chão. Se o jogador encostar nelas, recebe um dano, perdendo uma vida. A *sprite* é do pacote *Pixel Adventure 1*.

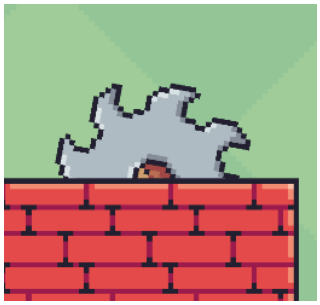
Figura 18 - Armadilhas *spikes*

Fonte: própria autora (2021)

4.5.3.3 Serras

Outro tipo de obstáculo, são serras giratórias que podem ou não se mover de um lado para outro. Assim como as *spikes*, elas dão dano quando são tocadas. Ambas estão destacadas nas figuras 19 e 20. As *sprites* também são do pacote *Pixel Adventure 1*.

Figura 19 - Exemplo de uma serra estática



Fonte: própria autora (2021)

Figura 20 - Exemplo de serra móvel

Fonte: própria autora (2021)

4.5.3.4 Plataformas

As plataformas, como mostra a figura 21, servem como um modo de transporte. Elas se movimentam de um determinado ponto até outro, repetidamente. As *sprites* são do pacote *Pixel Adventure 1*.

Figura 21 - Plataforma de transporte

Fonte: própria autora (2021)

4.5.3.5 *Checkpoint*

Um *checkpoint* marca o fim de uma fase. Ao chegar nele, o jogador conclui a fase atual e passa para a próxima. A figura 22 ilustra a bandeira de *checkpoint*. A *sprite* é do pacote *Pixel Adventure 1*.

Figura 22 - Bandeira de *checkpoint*

Fonte: própria autora (2021)

4.6 ELEMENTOS DO JOGO

4.6.1 Personagens

Frame é o personagem principal, ilustrado na figura 23, e sua tarefa é coletar frutas ao longo das fases, sempre desviando de objetos e inimigos que aparecerem pelo caminho. Nas figuras 24, 25 e 26 estão ilustrados, respectivamente, os inimigos: a galinha, o cogumelo e o camaleão. O jogador perde uma vida quando é atingido ou encosta em um inimigo. A galinha persegue o jogador quando o detecta em seu campo de visão. O camaleão, por sua vez, não persegue o jogador, mas o ataca quando este entra em seu campo de visão, como mostra a figura 27. O cogumelo é um inimigo passivo, não ataca o jogador, mas dá dano caso o jogador o toque. As *sprites* são dos pacotes *Pixel Adventure 1* e *Pixel Adventure 2*.

Figura 23 - Frame, o personagem principal

Fonte: própria autora (2021)

Figura 24 - Galinha correndo

Fonte: própria autora (2021)

Figura 25 - Cogumelo

Fonte: própria autora (2021)

Figura 26 - Camaleão parado

Fonte: própria autora (2021)

Figura 27 - Camaleão atacando

Fonte: própria autora (2021)

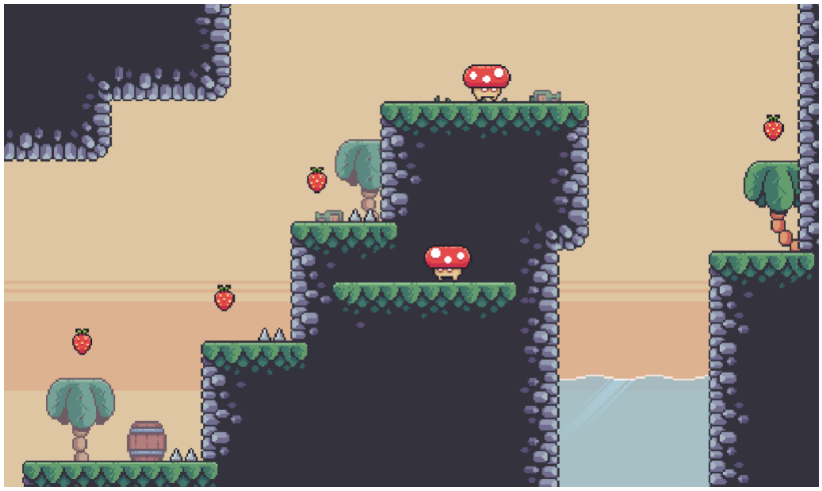
4.6.2 Fases e níveis

Atualmente, *Pixel Frame* conta com cinco fases, e não há limites estipulados para a quantidade de fases que o jogo pode ter. Nenhuma fruta ao longo da fase é gerada randomicamente. As fases foram construídas com *sprites* dos pacotes *Pixel Adventure 1*, *Treasure Hunters* e *Kings and Pigs*. Nas figuras 28 e 29, estão ilustradas a primeira e a quinta fase.

Figura 28 - Primeira fase

Fonte: própria autora (2021)

Figura 29 - Quinta fase



Fonte: própria autora (2021)

4.6.3 Interface

Os elementos seguem o mesmo estilo *pixel art* do jogo. A interface foi construída com *assets* do pacote *Treasure Hunters*, e o *cursor* do *mouse* é do pacote *MiniFX*. A figura 30 mostra os contadores de vida e de frutas coletadas. Nas figuras 31, 32, 33 e 34 estão ilustrados o *cursor* do *mouse*, menu principal, menu de pause e menu de opções.

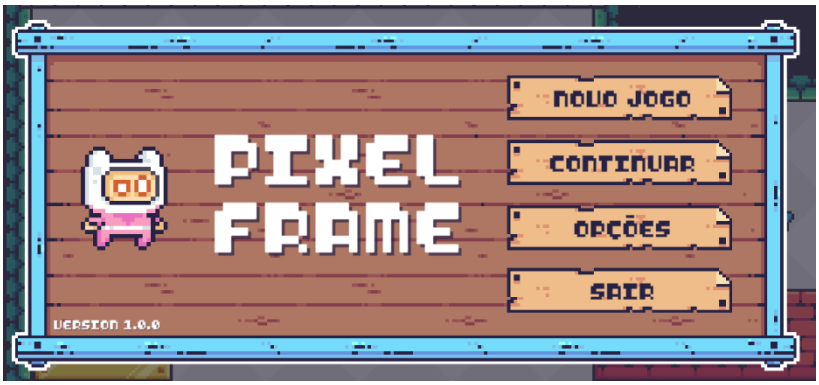
Figura 30 - Contadores de vidas e de frutas coletadas

Fonte: própria autora (2021)

Figura 31 - *Cursor* do *mouse*

Fonte: própria autora (2021)

Figura 32 - Interface do menu principal



Fonte: própria autora (2021)

Figura 33 - Interface do menu de pause

Fonte: própria autora (2021)

Figura 34 - Interface do menu de configurações

Fonte: própria autora (2021)

4.7 RESULTADOS E DISCUSSÃO

Durante a fase de planejamento geral sobre os aspectos do jogo, o gênero e o estilo foram definidos com o intuito de selecionar uma combinação que propiciasse a criação de um protótipo funcional no menor tempo possível, para validar as funcionalidades e os requisitos, e focar no aperfeiçoamento do fator diversão. O gênero plataforma e o estilo *pixel art* permitiram

que o desenvolvimento fosse mais veloz. Um jogo plataforma apresenta conceitos menos complexos de serem implementados, e o estilo *pixel art* facilita a customização de *assets* disponíveis. Requisitos foram levantados juntamente com algumas características primordiais para a confecção do *backlog*.

Na fase de desenvolvimento, o foco principal foi construir fases temporárias e finalizar a movimentação do jogador. Assim que alguns problemas relacionados à movimentação foram corrigidos e melhorias foram implementadas, o foco foi colocado nos obstáculos, nas mecânicas de interação, conclusão de fase e evento de morte do jogador. Durante todo o processo de desenvolvimento do projeto, houve uma mescla de algumas características das metodologias *Game Scrum* e XGD que se mostraram mais adequadas ao escopo do projeto. Assim que as funcionalidades básicas foram finalizadas, o primeiro protótipo completamente funcional ficou pronto.

Com essa primeira versão do jogo concluída, todas as funcionalidades estabelecidas no planejamento do projeto foram satisfeitas, cumprindo com os objetivos e o escopo deste projeto. A partir deste ponto, um melhor polimento do jogo e do fator diversão está sendo realizado. O *backlog* atual conta com melhorias a serem implementadas em versões futuras que lapidarão cada vez mais a jogabilidade e a diversão de *Pixel Frame*.

4.8 CONSIDERAÇÕES FINAIS

Uma mescla de algumas características do *Game Scrum* e do XGD se encaixaram satisfatoriamente para esse projeto de pequena proporção. A divisão em três fases, proposta pelo *Game Scrum*, viabilizou uma organização e uma modelagem melhor do projeto, simplificando o *backlog* para que as funcionalidades mais essenciais fossem o foco principal. Isso ocorreu porque os requisitos, funcionalidades, identidade visual e outros aspectos foram esculpidos de forma que o desenvolvimento pudesse ser encaixado no tempo limitado para criação dessa primeira versão do jogo.

Dado que esse projeto não conta com uma equipe maior com envolvimento de outras áreas, a separação em *sprints* não foi necessária. A fase de produção contou com implementações contínuas para a montagem do protótipo, porém sem a implementação de testes automatizados, pois não fazem parte dos objetivos e escopo do projeto.

Não iniciar a documentação do jogo durante a fase de produção, como sugere o XGD, deixou o desenvolvimento menos saturado e mais focado no amadurecimento de aspectos importantes do jogo como detalhes que aprimoram os fatores diversão e desafio. O *Game*

Design Document – documento que engloba todas as particularidades e aspectos de um jogo – começou a ser escrito na fase de pós-produção, quando a primeira versão foi finalizada. A pós-produção não indica necessariamente o fim do desenvolvimento do jogo, mas sim que a primeira versão dele está pronta para ser testada por um público selecionado, a fim de coletar pareceres sobre o jogo como um todo.

As ferramentas escolhidas atenderam às necessidades do projeto, possibilitando que os objetivos propostos fossem alcançados. Conclui-se que é completamente possível produzir um jogo de pequena escala, completo e divertido com tais ferramentas apresentadas, mesmo sem uma equipe completa com especialistas. Para este projeto, a fim de diminuir a complexidade, *Pixel Frame* foi projetado somente para computadores. Entretanto, também é possível construir jogos para outras plataformas como Android e iOS com o Unity.

À medida que o jogo for crescendo e o projeto amadurecendo, certamente mais ferramentas e procedimentos serão necessários para cumprir com tais novos objetivos, versões para outras plataformas serão criadas, a documentação ficará mais verbosa e completa, e sugestões de aperfeiçoamentos serão consideradas para implementação em versões futuras. *Pixel Frame* é um projeto de código aberto, livre para receber contribuições e adaptações, e pode ser acessado através do link: https://github.com/darayve/pixel-frame.

REFERÊNCIAS BIBLIOGRÁFICAS

BUSINESSWIRE. **Global Mobile Gaming Market Report 202**: Worth $98 Billion in 2020, the Mobile Gaming Market is Forecast to Grow to $272 Billion by 2030 - ResearchAndMarkets.com. 2021. Disponível em: <https://www.businesswire.com/news/home/20210625005180/en/Global-Mobile-Gaming-Market-Report-202-Worth-98-Billion-in-2020-the-Mobile-Gaming-Market-is-Forecast-to-Grow-to-272-Billion-by-2030---ResearchAndMarkets.com>. Acesso em: 07 set. 2021.

CLEMENT, J. **COVID-19 impact on the gaming industry worldwide** - statistics & facts. 2021. Disponível em: <https://www.statista.com/topics/8016/covid-19-impact-on-the-gaming-industry-worldwide/>. Acesso em: 15 set. 2021.

DEMACHY, T. **Extreme Game Development**: Right on Time, Every Time. 2003. Disponível em: <https://www.gamedeveloper.com/production/extreme-game-development-right-on-time-every-time/>. Acesso em: 25 jun. 2020.

DUSTYROOM. **FREE Casual Game SFX Pack**. 2019. Disponível em: <https://assetstore.unity.com/packages/audio/sound-fx/free-casual-game-sfx-pack-54116>. Acesso em: 07 nov. 2021.

FROG, P. **Pixel Adventure 1**. 2019. Disponível em: <https://assetstore.unity.com/packages/2d/characters/pixel-adventure-1-155360>. Acesso em: 07 nov. 2021.

FROG, P. **Pixel Adventure 2**. 2019. Disponível em: <https://assetstore.unity.com/packages/2d/characters/pixel-adventure-2-155418>. Acesso em: 07 nov. 2021.

FROG, P. **Treasure Hunters**. Disponível em: <https://pixelfrog-assets.itch.io/treasure-hunters>. Acesso em: 07 nov. 2021.

FROG, P. **Kings and Pigs**. Disponível em: <https://pixelfrog-assets.itch.io/kings-and-pigs>. Acesso em: 07 nov. 2021.

GODOY, A; BARBOSA, E. F. **Game-Scrum**: An Approach to Agile Game Development. 2010. Disponível em: <http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.655.44>. Acesso em: 25 jun. 2020.

GRAFXKID. **Mini FX, Items & UI**. 2020. Disponível em: <https://grafxkid.itch.io/mini-fx-items-ui>. Acesso em: 07 nov. 2021.

GUEDES, M. **Afinal, o que é TDD?**. 2019. Disponível em: <https://www.treinaweb.com.br/blog/afinal-o-que-e-tdd/>. Acesso em: 20 nov. 2020.

GULARTE, D. **Castlevania (Konami, 1986)**. 2010. Disponível em: <https://bojoga.com.br/retroplay/analises-de-jogos/nes/castlevania-konami-1986/>. Acesso em: 31 out. 2021.

GWRITERSTUDIO. **8Bit Music Album - 051321**. 2021. Disponível em: <https://assetstore.unity.com/packages/audio/music/8bit-music-album-051321-196147>. Acesso em: 07 nov. 2021.

HERN, H. **Fun factor for game developers**. 2002. Disponível em: <https://www.gamedev.net/tutorials/game-design/game-design-and-theory/fun-factor-for-game-developers-r1828/>. Acesso em: 20 nov. 2020.

JACOB. **What Are Sprites And How They Work In Games?**. 2020. Disponível em: <https://gamingshift.com/sprites-in-games/>. Acesso em: 21 nov. 2020.

MARQUES, M. **Game Engines:** conceitos e aplicações no desenvolvimento de jogos digitais. 2020. Disponível em: <https://warpzone.me/game-engines-conceitos-e-aplicacoes-no-desenvolvimento-de-jogos-digitais/>. Acesso em: 20 nov. 2020.

NEWZOO. **Newzoo Global Games Market Report 2021 | Free Version.** 2021. Disponível em: <https://newzoo.com/insights/trend-reports/newzoo-global-games-market-report-2021-free-version/>. Acesso em: 15 set. 2021.

NINTENDO. **The official home of Super Mario™** - History. Disponível em: <https://mario.nintendo.com/history/>. Acesso em: 20 nov. 2020.

OMS. **Covid-19**: OMS divulga guia com cuidados para saúde mental durante pandemia. 2020. Disponível em: <https://news.un.org/pt/story/2020/03/1707792/>. Acesso em: 28 out. 2021.

REIS, T. **CAGR:** o que é a Taxa de Crescimento Anual Composta?. 2019. Disponível em: <https://www.suno.com.br/artigos/cagr/>. Acesso em: 15 set. 2021.

SCHOFIELD, B. **Embracing Fun:** Why Extreme Programming is Great for Game Development. 2007. Disponível em: <https://www.gamedeveloper.com/design/embracing-fun-why-extreme-programming-is-great-for-game-development/>. Acesso em: 19 nov. 2020.

SOMMERVILLE, I. **Engenharia de Software**. 9. ed. São Paulo: Pearson Education, 2011.

STEAM. **Ristar™ no Steam**. 2010. Disponível em: <https://store.steampowered.com/app/34312/Ristar/>. Acesso em: 31 out. 2021.

UNITY. **Common types of assets**. 2020. Disponível em: <https://docs.unity3d.com/Manual/AssetTypes.html>. Acesso em: 20 nov. 2020.

UNITY. **Creating Gameplay**. 2020. Disponível em: <https://docs.unity3d.com/Manual/CreatingGameplay.html>. Acesso em: 20 nov. 2020.

UNITY. **Gameplay in 2D**. 2020. Disponível em: <https://docs.unity3d.com/Manual/Overview2D.html>. Acesso em: 20 nov. 2020.

UNITY. **Unity - Manual:** System requirements for Unity 2020 LTS. 2021. Disponível em: <https://docs.unity3d.com/Manual/system-requirements.html#desktop/>. Acesso em: 31 out. 2021.

UNITY. **Welcome to Unity**. 2020. Disponível em: <https://unity.com/our-company>. Acesso em: 20 nov. 2020.

WELLS, D. **Extreme Programming:** A gentle introduction. 2013. Disponível em: <http://www.extremeprogramming.org/>. Acesso em: 19 nov. 2020.