Etapas de desenvolvimento: protótipo, programação, testes e implementação

Ciclo de produção de programação

Continuamos a criação do jogo com referência no plano de desenvolvimento elaborado a partir do GDD e do TDD. Relembrando o conteúdo deste plano, temos, segundo Chandler (2012), o trabalho que deve ser feito, a ordem em que deve ser realizado, quem o fará e quando deve ser concluído, o orçamento e as etapas de desenvolvimento (alfa, beta, candidata etc.).

Para mais detalhes, recorra ao material do curso sobre as metodologias de desenvolvimento.

O desenvolvimento do jogo digital, em sua fase de planejamento, requer pensar nos ciclos de produção para as equipes de arte, *design*, programação e qualidade. Em relação à programação, segue-se uma linha de desenvolvimento para atender aos requisitos levantados.

A partir das atividades de levantamento do que constará no jogo, elabora-se um cronograma de atividades e estima-se o esforço necessário para iniciar o desenvolvimento. Um ciclo de vida de desenvolvimento requer a passagem pelas seguintes etapas: elencar o que será feito, estimar o tempo que levará, programar, testar, integrar e implantar. Por vezes, no desenvolvimento de *games* alguma jogabilidade não fica clara para a equipe. Desta forma, cria-se um protótipo na pré-produção para que se experimente como ficou uma certa jogabilidade ou ferramenta e então passa-se para a

produção.

A seguir, veremos o ciclo de produção de programação de jogos digitais mais detalhadamente.

Protótipo

O que é um protótipo? Já responderemos, mas primeiro vamos pensar: por que elaborar um protótipo? Imagine uma equipe em reunião de *brainstorming* pensando na jogabilidade. Após uma hora de discussão, chegam a um mecanismo, mas a equipe ainda não tem ideia se esse mecanismo é divertido.

Para isso, elabora-se um protótipo para testar a jogabilidade e refiná-la, caso seja necessário.

Agora, imagine que um estúdio iniciou o desenvolvimento de um novo *game* de entretenimento, porém não obteve financiamento para a ideia e está atrás de patrocinador. Um dos recursos utilizados para vender a ideia é o protótipo. Ainda, no caso apresentado por Rogers (2013), de um estúdio que está desenvolvendo um jogo solicitado por um *publisher*, é necessário dar o *feedback* (o retorno) a este, para que ele avalie se o desenvolvimento está conforme o solicitado.

Você já deve ter ideia do que é um protótipo. Se compreendeu o protótipo como uma forma de capturar a mecânica do jogo, com uma qualidade que permita a sua avaliação, você está certo. Conforme Chandler (2012), um protótipo, no desenvolvimento de jogos, é uma proposta de mecanismo ou ideia de jogo. O mesmo autor destaca que, no processo de desenvolvimento de *software*, a prototipagem é considerada uma etapa prévia ao desenvolvimento, pois auxilia na compreensão do que deve ser feito. Os protótipos são utilizados na comunicação com o cliente e tornam a ideia do que fazer mais clara para a equipe e para ele.

Mas como utilizar o protótipo? O que fazer com o resultado do protótipo? Novak (2012) destaca que o protótipo deve ser utilizado, principalmente, para testar a jogabilidade. Então, quem faz o teste? A equipe de desenvolvimento pode realizar o teste, mas o mais interessante, argumenta Fullerton, citada por Novak (2012), é que seja criado um grupo de teste com jogadores do público-alvo do *game*, pois a contribuição gerada será surpreendente. Não necessariamente será bem recebida, mas com certeza

direcionará o desenvolvimento para atender aos jogadores que são foco do jogo. As autoras citadas recomendam que os protótipos sejam feitos cedo, que os testes sejam realizados sempre, que os objetivos sejam claros e que os retornos (*feedback*) sejam incorporados ao projeto.

Como desenvolver um protótipo? Os protótipos podem ser físicos ou digitais, ou seja, de baixa ou alta fidelidade, respectivamente. Você entendeu certo, podemos ter protótipos não digitais de jogos digitais. Conceba mentalmente um jogo de tiro em primeira pessoa (FPS), como Unreal Tournent, Doom, Quake, Battlefield 1942, os quais envolvem um mecanismo simples como um jogador que disfere tiros contra outro personagem em um cenário. Seria possível representar uma jogabilidade como essa em um protótipo físico?

Fullerton (2004) apresenta um exemplo de como representar fisicamente uma jogabilidade de FPS. Para este caso, elabora-se um tabuleiro de papel, com a subdivisão dos caminhos possíveis (hexágonos desenhados no papel), com os obstáculos representados com folhas de caderno e clips, e posicionando bonequinhos de plástico nos possíveis lugares do cenário para representar o *player* e os inimigos. A mecânica é simulada com dados e objetos para representar as jogadas e a pontuação, tal qual pode ser visualizado na imagem abaixo.

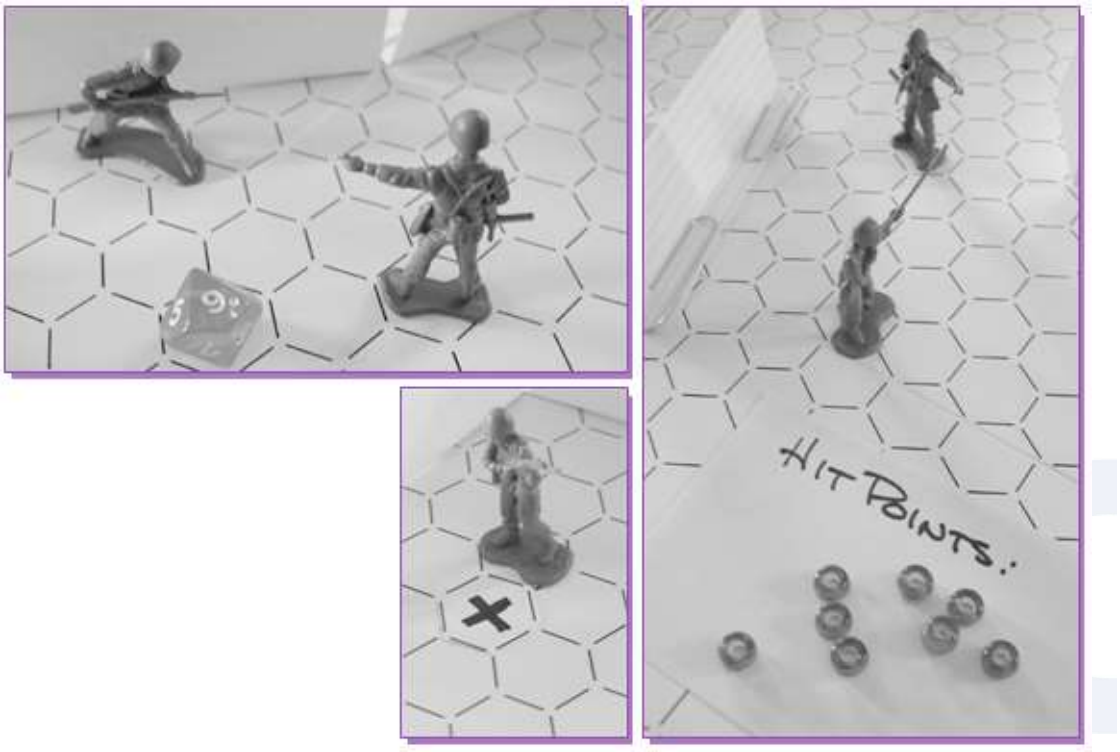

Figura 1 – Exemplo de protótipo físico para um jogo de FPS

Fonte: Fullerton (2004).

A mecânica é exemplificada da seguinte maneira: o personagem move-se três espaços e atira; o personagem inimigo move-se 2 espaços e atira. A quantidade de espaços é determinada pelo valor apresentado ao jogar o dado.

Os protótipos de alta fidelidade são versões digitais jogáveis que apresentam elementos do cenário e personagens levemente caracterizados, isto é, contendo algumas texturas e iluminação.

SAIBA +

Dica de vídeos: procure no YouTube por protótipos de jogos e pelo “Umbra”, um protótipo de alta fidelidade de um jogo com lançamento em 2016.

Busca-se identificar novos requisitos com protótipos exploratórios, e validar requisitos com os experimentais. Dentro destas finalidades, são prototipadas jogabilidades, sistemas de animação, ferramentas de engenharia, *scritps* de automação da linha de produção da ferramenta, áudio, jogabilidades de tiro, inteligências artificiais dos personagens controlados por computador, automatização da substituição das animações dentro da ferramenta.

Neste processo de experimentação, algumas criações podem ser descartadas, gerando uma sensação de perda de tempo, mas no fundo é o contrário, pois a equipe adianta-se em relação ao que funciona antes de colocar o jogo em produção, eliminando o risco de descobrir que uma jogabilidade não funciona quando o prazo de lançamento estiver próximo. Logo, a realização de protótipos constitui um investimento, mitigando riscos.

Fullerton, citada por Novak (2012), reforça que uma parte da equipe deve ficar focada no desenvolvimento dos protótipos na pré-produção. Após a validação, deve-se colocar toda equipe para desenvolver a jogabilidade, passando para a etapa de produção. Logo, o protótipo seguirá para a programação.

Dica de pesquisa científica sobre protótipo: busque sobre avaliação heurística para protótipos de jogos digitais.

Programação

A programação de jogos digitais envolve a codificação de uma série de elementos que não constam na maioria dos sistemas computacionais, como a integração de animações, áudio e imagem, além dos movimentos do personagem. Um jogo digital pode ser dividido, conforme McShaffry (2013), na seguinte arquitetura: camada de aplicação, camada lógica e visões do jogo.

Conhecendo a arquitetura de um jogo, isto é, os elementos que o compõem, percebemos que programar um jogo é um trabalho difícil. A camada de aplicação é composta pela programação dos dispositivos de entradas, isto é, controles do jogo, pelo gerenciamento dos arquivos (*assets*), pelo gerenciamento da memória e pelo gerenciamento do tempo do jogo. Também é composta pela programação da interface do jogo com o sistema operacional, ou seja, a linguagem, as bibliotecas (DLLs), *threads* (para o processamento paralelo) e rede. Por fim, nesta camada, ainda são programadas as bibliotecas centrais do jogo, o *loop* principal, o início e o fim do jogo e a linha do tempo do jogo.

A camada lógica contém os estados e as estruturas de dados do jogo, a física do mundo do *game*, os eventos, o gerenciamento de processos e o interpretador de comandos (para as linguagens *script*). As visões do jogo consistem nas visões do *player*, da inteligência artificial e de rede.

A codificação desses elementos necessita ser agendada conforme a prioridade de desenvolvimento. Alguns desses elementos serão abordados ao longo do curso e veremos a utilização de ferramentas como os SDKs, as bibliotecas físicas, as game engines etc.

Para atender a toda esta demanda, a etapa de programação deve contar com uma equipe composta, segundo Novak (2012) e Mcshaffry (2013), por, primeiramente, um diretor-técnico, que cria o *design* técnico do projeto e supervisiona a implementação ao longo do processo. Ele também seleciona as ferramentas, especifica o *hardware* necessário e os padrões de código. O time também é composto por um programador líder, que é responsável por supervisionar diariamente as atividades dos programadores. Ele realiza o projeto técnico de como será implementada uma característica do jogo (*feature*), como uma jogabilidade, ou como será implementada a conexão de rede para um jogo *multiplayer*.

A programação do jogo envolve ainda programadores de rede, dos gráficos, da *engine*, das ferramentas, do áudio, da física e da interface, além de um programador júnior, que adiciona pequenos elementos ao projeto do jogo.

Podemos dividir o código do jogo, de forma abstrata, entre o domínio e o suporte (por exemplo, o código do motor do jogo que dá suporte para o desenvolvimento de diferentes jogos de um mesmo tipo). Imagine que desenvolvemos um *game* de tiro em primeira pessoa para a Revolução Francesa de 1848. Então, desenvolvemos a física, a interação entre os personagens e o cenário. Quando formos desenvolver um jogo para a Revolução Farroupilha do Rio Grande do Sul, de 1845, não precisaremos programar a física do tiro, a interação dos personagens e do cenário, apenas mudamos o domínio do jogo, a arte, algumas interações distintas e a história. O motor de jogo Unreal UDK (saiba mais sobre a UDK no material sobre as bibliotecas) evoluiu, desta forma, para jogos de tiro, como o Unreal Tournament, mas agora aborda diferentes gêneros de *games*.

A etapa de programação se trata do projeto, da implementação e da integração das funcionalidades (*features*), das ferramentas de suporte e dos elementos do domínio do jogo. Após essa etapa, passa-se para uma equipe que executará os testes no código do jogo. Esta última etapa pode começar antes da etapa de programação, quando são planejados os testes que serão executados utilizando a metodologia de desenvolvimento dirigido por testes, ou depois. Vamos ver mais sobre a etapa de testes a seguir.

Testes

“Ebaaaa! Vamos jogar muuuiiiiitoooo!”

Parece que realizar testes em jogos deve ser divertido, mas Chandler (2012) ressalta que esta tarefa não é tão prazerosa quanto parece. Testar um *game* é uma atividade estressante, a busca por falhas em uma missão requer repetir uma série de vezes as mesmas atividades e, após algumas semanas, percebe-se que não é tão divertido testar.

Todavia, testes são essenciais em qualquer projeto de *software*. Em um jogo não é diferente: o impacto de um *game* com falhas pode ser desastroso para o jogador. Em uma situação hipotética, você está jogando um RPG, seu personagem está no nível 77 de 100 e ocorre uma falha no jogo, levando você de volta para o nível 1. Certamente, você não vai jogar até obter o nível 77 de novo. Neste caso, perdeu-se um jogador e ganhou-se um inimigo. Quantas horas foram desperdiçadas? Ao longo de todo o processo de desenvolvimento do jogo, devem ser realizados testes. Cada *feature* nova deve ser testada, por exemplo: a jogabilidade, a animação do jogo, a colisão física, as ferramentas e o jogo depois de pronto.

E quais são as causas das falhas? Ocorrem falhas em função de erros de integração de bibliotecas, de compiladores, de sistemas operacionais, de *drivers*, mas principalmente de erros em códigos escritos por diferentes programadores. Por exemplo, uma funcionalidade que é testada, separadamente, com um teste de unidade, e funciona; porém, ao integrar-se com as demais funcionalidades, ocorrem erros.

Para isso, mantém-se uma equipe de teste que planeja os testes, executa-os e faz o gerenciamento deles visando à qualidade no desenvolvimento do jogo. Esta é a equipe de garantia da qualidade (CHANDLER, 2012). A qualidade não trata apenas do jogo, mas do processo como um todo. Por exemplo, durante o desenvolvimento do jogo, ocorre uma série de *bugs* (defeitos), que são identificados pelos testadores e registrados; então é feito um levantamento periódico para verificar o grau de ocorrência de erros, e isso é melhorado junto com a equipe de programação.

Para exercer a atividade profissional no desenvolvimento de jogos, é importante que você domine algumas técnicas de teste. Temos os seguintes tipos de testes: caixa branca, caixa preta, teste de unidade, testes de integração e testes de jogabilidade (*playtest*). A estes últimos tipos de teste, Novak (2008) acrescenta a função de balanceamento da dificuldade. Vamos abordar estas técnicas de maneira aprofundada ao longo do curso.

Existe uma série de ferramentas disponíveis para realizar os testes. Os *debuggers* ajudam a encontrar erros no código. Hoje em dia, os erros de sintaxe de código são identificados nas IDE de programação automaticamente, bem como dependências de bibliotecas necessárias. O motor de jogo Unity 3D disponibiliza, gratuitamente, na sua loja de *assets*, uma ferramenta de testes de unidade, de integração, de componentes e de tempo de execução. Outro exemplo de ferramenta de testes são os *frameworks* de teste de unidade, como NUnit, o *framework* utilizado para as ferramentas da Microsoft .Net, como a linguagem de programação C#.

Mas como se dá o processo de teste ao longo do desenvolvimento do jogo digital? Desde a pré-produção, conforme Chandler (2012), o líder de garantia de qualidade deve participar indicando quais *features* do jogo gerarão desafios para a equipe de testes. O cronograma de testes deve ser sincronizado com o cronograma de liberações das versões do *game*, como a demo, como os executáveis para o *marketing* e como a liberação do código. Esse cronograma pode ser elaborado junto com o de desenvolvimento do jogo. Indicam-se as dependências para destacar o impacto das mudanças no processo de testes. Também é elaborado um plano de testes que deverá contemplar a verificação de todas as áreas do jogo. Tanto o cronograma quanto os planos de teste devem ser constantemente atualizados.

Considerando o nível 1 do nosso jogo Alien City, segue um exemplo de plano de teste. Vamos partir do conflito de abduções do jogo. Supomos que o nosso personagem principal (jogador), John Samson, está na praça e lá ele deve evitar uma abdução. Quando ocorre uma abdução, há um corte na cena e ela muda para uma animação em que desce uma luz da nave para capturar uma pessoa. Quando a pessoa está subindo, a cena volta para a câmera do jogador, que deve correr na direção da pessoa e executar o comando que faz Samson saltar para impedir a abdução. Neste momento, troca a câmera do *player* para uma animação com o salto e o salvamento, retornando logo em seguida para a câmera do jogador. O que deve ser testado, neste caso, são os cortes e os

carregamentos das cenas, o tempo de troca entre as cenas, a resposta ao controle para salvar a pessoa e a localização do jogador em relação à pessoa em abdução para que a ação de salvamento seja efetiva. Com esses requisitos em mente, elaboramos o seguinte plano de teste do tipo aprovado/reprovado para essa jogabilidade.

Jogabilidade de abdução	Requisitos	Aprovado/reprovado	Notas
Troca da cena: início abdução	A troca de cena deve ocorrer sem cortes.	Aprovada	
Troca da cena: pessoa sendo abduzida — visão do jogador	A troca da cena deve ocorrer sem cortes.	Reprovada	O tempo de troca da cena está demorando muito para começar.
Ação do jogador: salvamento	Acionada apenas se estiver dentro do raio de proximidade da pessoa sendo abduzida quando o jogador acionar o controle.	Aprovada	
Troca de cena: salvamento	Deve ocorrer sem cortes.	Aprovada	
Troca de cena: salvamento	Ao finalizar, deve retornar para a câmera do jogador.	NPT	Não pode ser testado, o jogo travou inesperadamente.
...			

O testador, após jogar e verificar os elementos do jogo conforme os requisitos identificados, informa se foi aprovado, reprovado ou se não foi possível testá-lo. No exemplo, vimos que o item “troca de cena: início abdução” atendeu ao requisito “troca sem cortes” e foi aprovado. A “troca de cena: pessoa sendo abduzida – visão do jogador” não atendeu ao requisito “troca sem cortes” e foi reprovada. A “ação do jogador: salvamento” atendeu ao requisito de proximidade da abdução, quando o controle fosse acionado e foi aprovada. A “troca de cena: salvamento” não pode ser testada, pois, conforme indicado na nota, o jogo travou inesperadamente antes de ocorrer a troca.

No plano de testes, tanto Chandler (2012 p. 360) quanto Novak (2008 p. 229) destacam que deve haver a classificação dos bugs, a descrição, os tipos de testes, o cronograma de

testes e a definição do *pipeline* (ferramentas de rastreamento e processo de relato), quem ficará responsável por realizar cada teste e pela correção dos *bugs*. No cronograma, deve estar definido o ciclo de testes, ou seja, quando ocorrerão os testes, em quais *builds* serão realizados e com qual intensidade.

Dica: veja no livro *Manual de produção de jogos digitais*, na página 361, um exemplo de plano de testes.

A etapa de teste evoluiu ao longo do tempo e ganhou a devida relevância no processo de desenvolvimento de jogos. Atualmente, este processo está bastante automatizado, com uma série de ferramentas integradas, fazendo com que esta parte do ciclo de produção de programação seja bastante eficiente, desde o planejamento de uma *feature* até a implantação do jogo. Abordaremos o que é e como ocorre a etapa de implantação a seguir.

Implantação

A implantação é o processo de gerar os executáveis do jogo em ambiente de testes, ou de produção.

Para Rabin (2012), esse processo é simples quando se trata de um projeto pequeno em que utilizamos um ambiente de desenvolvimento integrado (IDE – *Integrated Development Environment*) que gera uma *build* (versão executável) do sistema. No entanto, para jogos, que são sistemas computacionais complexos, gerar uma *build* não é um processo simples, pois envolve uma série de arquivos de código, bibliotecas, *assets* etc. Os motores de jogo facilitam esse trabalho. A Unity 3D, por exemplo, permite gerar executáveis de maneira bastante simples e para diferentes plataformas. Em poucos passos, é possível gerar uma *build*, selecionando as cenas que serão incluídas, enquanto o motor seleciona automaticamente arquivos e *assets* que serão agregados.

No entanto, para projetos grandes, argumenta Rabin (2012), é preciso, ainda assim, realizar certas medidas que dão suporte para esta etapa de desenvolvimento. Utilizar um controle de versão integrada com uma ferramenta de geração do executável e de teste é uma das tarefas essenciais em projetos. Para citar um exemplo, voltamos à Unity 3D. Ela possui um recurso *on-line* que se chama Unity Cloud Build, que integramos ao nosso repositório de arquivos e com isso geramos automaticamente uma *build* que fica disponível na *cloud* para a equipe. Este recurso dá suporte para a integração contínua e para a colaboração com a equipe à medida que a *build* já fica disponível para todos. Você deve estar pensando que a equipe de teste pode pegar esta versão da *cloud* e realizar os testes nela. Isso mesmo! Essa ferramenta possibilita a integração com a equipe de qualidade que realizará os *playtests* no jogo.

A integração contínua (e a entrega contínua) visa a mitigar o efeito de erros de desenvolvimento, proporcionando à equipe de qualidade a possibilidade de encontrar os defeitos com os testes ao longo do desenvolvimento, eliminando, com isso, as possíveis falhas no jogo. Esta metodologia é viabilizada pela Unity Cloud Build, por exemplo.

Você já conseguiu perceber a vantagem de gerar as *builds* ao longo do ciclo de programação? O processo de implantação, como citado, permite a identificação de falhas o quanto antes. A automatização da geração da *build* torna o processo mais fluido e flexível na medida em que ela pode ser gerada sob demanda das equipes de *marketing* ou de garantia da qualidade. A automatização torna eficaz o processo de teste quando for necessário intensificar a criação dos executáveis, como cita Chandler (2012), de duas a três vezes por semana para a primeira versão jogável e diariamente para a versão alfa.

A definição das ferramentas e do processo de geração da *build* deve ser planejado na fase de pré-produção do jogo, juntamente com um cronograma. A implementação do processo deve ser realizada o quanto antes, destaca Chandler (2012 p. 320).

Por fim, a criação da *build* é a uma etapa importante do ciclo de produção de programação, essencial para verificar a qualidade e para lançar o jogo em sua versão final. A identificação das falhas na versão executável ajuda a corrigir os erros cometidos na especificação e na programação, além dos não detectados nos testes. Esse ciclo de programação é essencial para o desenvolvimento de jogos com qualidade e eficiência.

Bibliografia

CHANDLER, Heather Maxwell. **Manual de produção de jogos digitais**. 2ª ed. Porto Alegre: Bookman, 2012.

FULLERTON, Tracy, *et al*. **Game Design Workshop**: Designing, prototyping and playtesting Games. USA: CMP Books, 2004.

MCSHAFFRY, Mike; GRAHAM, David. **Game Coding Complete**. 4ª ed. Boston: Cengage Learning, 2013.

NOVAK, Jeannie. **Game Development Essentials**. 3ª ed. Canada: Cengage Learning, 2012.

NOVAK, Jeannie; HIGHT, John. **Game Development Essentials**: Game Project Management. New York: Cengage Learning, 2008.

RABIN, Steve. **Introdução ao Desenvolvimento de Games**: Programação – técnica, linguagem e arquitetura. São Paulo: Cengage Learning, 2012

ROGERS, Scott. **Level UP**. Blucher, 2013.