Introdução a Jogos Digitais

Aula 09 - Motores de Jogos I

Apresentação

Olá!

Hoje teremos uma aula mais técnica na área de computação! Calma, não entraremos em detalhes confusos nem nada do tipo, mas falaremos um pouco sobre como é um jogo do ponto de vista da programação. Vocês já deram seus primeiros passos na disciplina de programação estruturada, mas nem se preocupem, pois não iremos sair codificando algo avançado, apenas refletiremos alguns pontos para que vocês entendam como o desenvolvimento de um jogo não é uma tarefa simples.

Preparados? Então vamos lá!

Objetivos

- Entender a arquitetura de um jogo;
- Conhecer os vários aspectos de implementação de um jogo;
- Definir o que é um motor de jogo e sua importância para o desenvolvimento de jogos.

1 - Programando um Jogo

Vamos começar a programar?

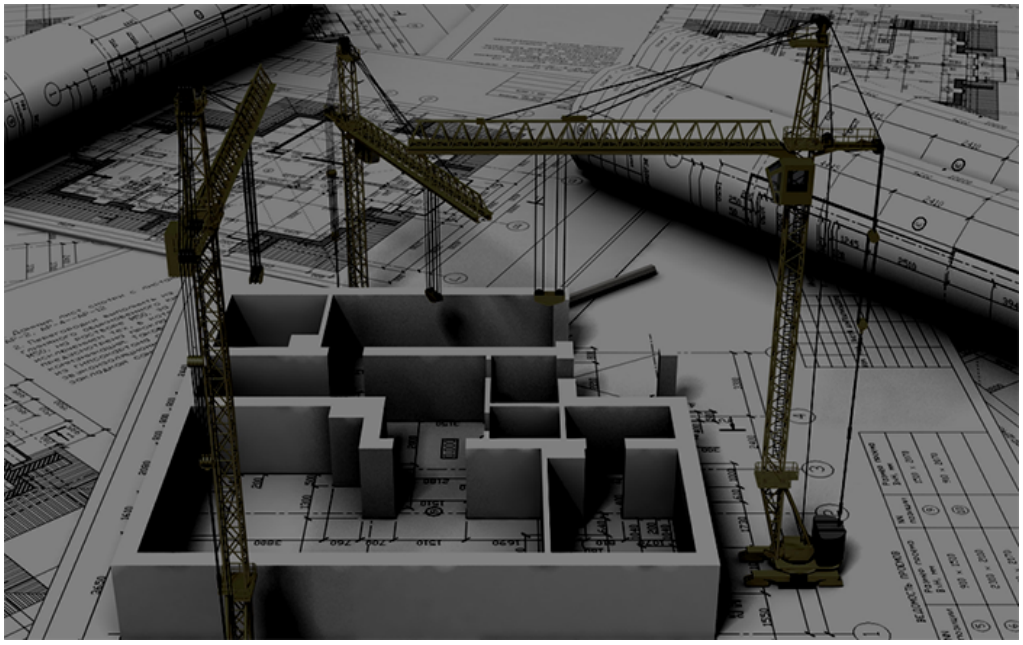
Figura 01 - Mas já, professor??

Calma, calma.

Vocês estão dando os seus primeiros passos na programação dentro do Curso Técnico (ou será que já chegaram sabendo?) e já estudaram várias estruturas que são usadas na construção de programas, como os laços de repetição, os condicionais, as variáveis, as funções, etc. Um jogo, como qualquer outro programa, também é construído usando essas estruturas, porém com uma complexidade mais alta. Isso acontece porque um jogo normalmente é composto por vários sistemas, pedaços maiores de código que compõem funcionalidades específicas operando de forma integrada dentro de uma arquitetura definida para o projeto.

A ideia é parecida com a arquitetura tradicional de casas que você conhece. Um projeto de arquitetura de um imóvel serve para dar uma ideia de como os elementos serão utilizados na sua construção, como os cômodos serão distribuídos na área do terreno, como eles serão interligados de forma a facilitar a circulação das pessoas e a ventilação da casa, qual a função de cada cômodo e o que é preciso na construção para que cada um possa desempenhá-la (por exemplo, não pode esquecer o encanamento no banheiro). A arquitetura de um software tem o mesmo propósito, pois a partir dela definiremos quais tipos de módulos ou componentes serão necessários para que o programa funcione, como eles irão se comunicar/trocar informações ao longo da execução do programa, quais tipos de padrões de código existem para realizar determinadas tarefas, e por aí vai.

Figura 02 - Assim como uma casa, um programa de computador também precisa de um bom planejamento antes de ser construído!



Dito tudo isso, não existe uma arquitetura padrão para jogos, normalmente fica a critério da equipe de desenvolvimento escolher ou criar, mas com objetivos a serem considerados: facilitar a programação do jogo, aumentar o desempenho/eficiência da aplicação, tornar mais fácil o desenvolvimento para dispositivos móveis, prover uma interface mais intuitiva para a equipe de artes, permitir uma maior flexibilidade na execução de alterações e testes, etc. Mesmo assim, é comum observar a divisão do jogo nas seguintes partes:

- **Operacional**: relaciona-se a detalhes da plataforma/sistema em que o jogo roda, e implementa aspectos mais técnicos de como o jogo acessa o recurso computacional para executar. Quando essa parte está ruim, o jogador costuma gritar:

    - Está muito lento esse jogo!

    - O controle não responde direito!

    - Travou na hora que eu ia derrotar o chefão!

- **Lógica do jogo**: trata-se do jogo em si, representa o conjunto de regras e elementos que definem o estado e o ciclo do jogo. Quando está ruim, o jogador grita:

    - Esse jogo não presta, é muito chato!

    - Não acredito que aconteceu isso com meu personagem!

    - Como eu ia adivinhar que tinha de pegar esse item na primeira fase para poder zerar o jogo?

- **Visualização**: relaciona-se à apresentação e renderização do jogo (desenho da tela do jogo). Quando essa parte está ruim, o jogador grita:

    - Que jogo feio!

    - Cadê o inimigo que eu estava atacando? Desapareceu do nada, mas continua tirando minha energia!

    - Será que essas partes faltando na tela são intencionais?

Para não dar um nó na cabeça de vocês, agora faremos uma abordagem em alto nível da programação de jogos. Vocês terão disciplinas mais à frente no curso (logo duas, Motores de Jogos I e II) que entrarão em mais detalhes nos aspectos técnicos. Dando início à nossa discussão, vamos primeiro pensar em um jogo e como ele funciona. Que tal usarmos o Pacman como exemplo? Sim, eu gosto de Pacman!

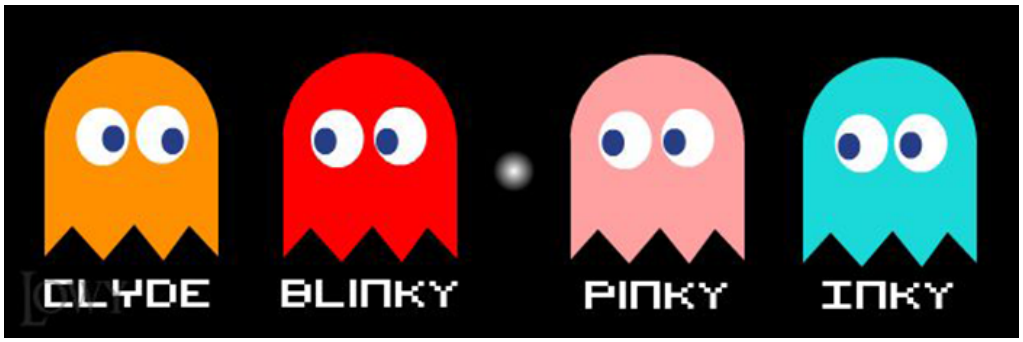
Além disso, ele é um jogo simples de entender e possui vários elementos que enriquecem a nossa discussão. E dá para experimentar de graça, é só ir no Google e jogar online mesmo!

Figura 03 - Pacman é um clássico que possui elementos usados até hoje nos jogos digitais.

Fonte: https://spritted.com/pt/news/pacman-o-eterno-legado. Acesso em: 26 ago. 2016.

No jogo do Pacman, você controla uma bolinha amarela que tem como objetivo comer todos os pontinhos do mapa, sem deixar que fantasmas coloridos lhe capturem. Uma aventura, no mínimo, mitológica! Para mover o Pacman, o jogador deve usar as teclas de direcionais (ou outro esquema que ele escolher) e movimentá-lo para cima/baixo e de um lado para o outro. O mapa do jogo é um labirinto que consiste em vários caminhos delimitados por paredes, e o Pacman não pode atravessar essas paredes (curiosamente, nem os fantasmas, mas pode ser só preguiça deles...). Caso os fantasmas peguem o Pacman, ele perde uma "vida" ou tentativa para finalizar o jogo (o jogador usualmente começa com três tentativas). Existem pontinhos especiais maiores que, ao serem comidos, dão um poder extra ao Pacman e faz com que ele possa comer os fantasmas, derrotando-os temporariamente. Cada fantasma possui um padrão de movimento próprio e o jogador deve ter cuidado e atenção para não ser encurralado por eles.

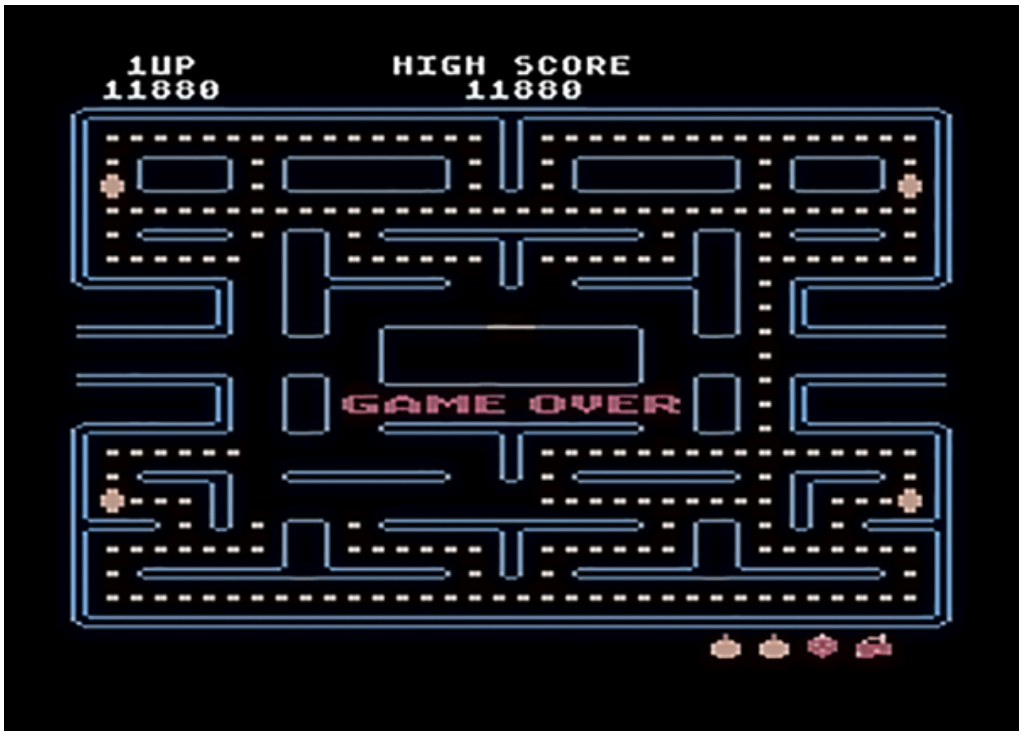
Figura 04 - Esses fantasminhas são uma turminha da pesada!



Fonte: http://www.garotasgeeks.com/5-coisas-que-voce-nao-sabia-sobre-o-pac-man/. Acesso em: 26 ago. 2016.

Essa é uma descrição curta, porém que dá uma noção geral dos vários elementos do jogo. Vamos analisar alguns pontos: primeiro, existe uma diferença entre o Pacman estar “energizado” e poder derrotar fantasmas, e o modo normal. Essa diferença não afeta apenas o confronto com os monstros, mas até a forma como eles se movimentam (eles fogem do jogador), ou seja, existem estados diferentes dentro do jogo que alteram o modo como os elementos se comportam ou se apresentam para o jogador. Dizemos que esse conjunto de atributos é o **estado do jogo**, e para que tudo funcione corretamente, esse estado precisa ser mantido constantemente atualizado. Em nível de programação isso significa manter um conjunto de variáveis e estruturas com informações sobre os vários elementos do jogo, e executar ações na medida em que o estado se transforma. Por exemplo, se todas as bolinhas presentes no mapa acabarem, o jogo precisa saber dessa ocorrência e que o jogador venceu a partida, exibindo informações na tela e perguntando se ele deseja jogar novamente ou sair. Se o Pacman for pego por um fantasma e não tiver mais tentativas, a temida tela de *Game Over* será exibida para o jogador!

Figura 05 - O estado do jogo permite que o desenvolvedor defina o resultado de cada interação e o que deve mudar na medida em que o jogador vai progredindo no jogo. E esse estado aí da figura todo jogador quer evitar!



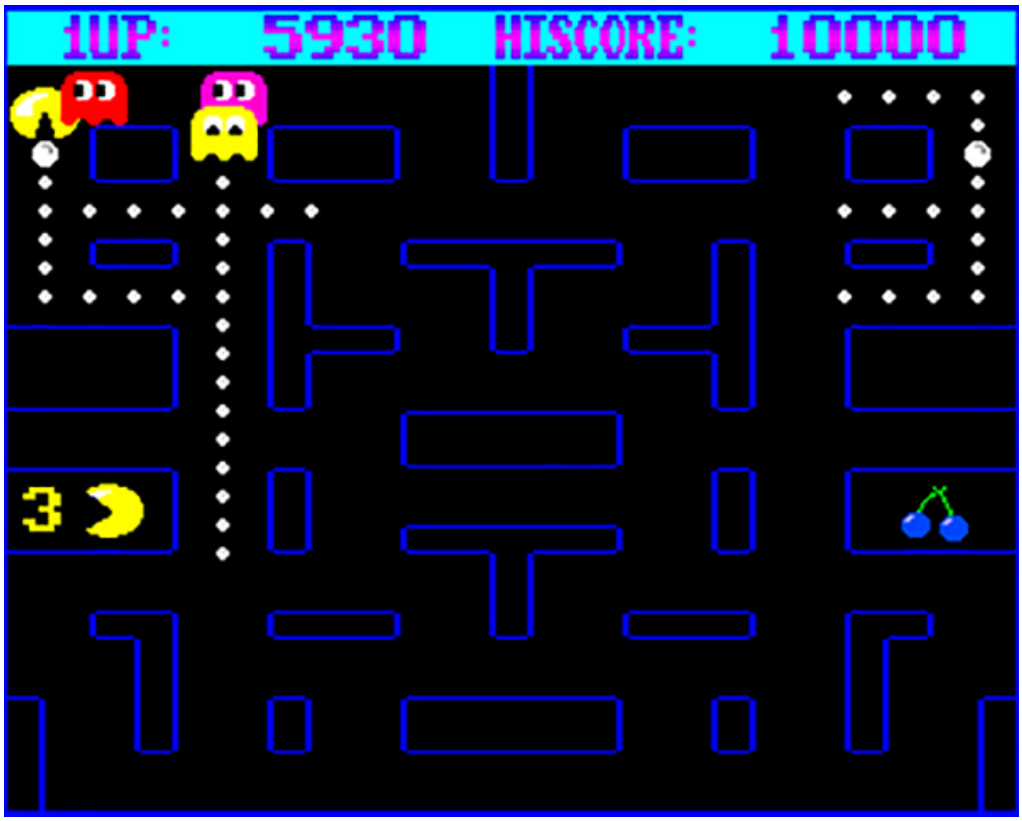
Fonte: http://www.mobygames.com/game/atari-8-bit/pac-man/screenshots/gameShotId,149590/. Acesso em: 26 ago. 2016.

Outro ponto importante é que o jogo ocorre em um **ciclo de ações**: o jogador vê o mapa, executa um comando, esse comando é processado pelo jogo, o estado do jogo é atualizado e apresentado ao jogador através do redesenho da tela. Esse ciclo se repete até que o jogador encerre a partida (ou se você jogar LOL até o cliente dá um crash naquela partida que você ia carregar com nota S...). Na programação, isso corresponderia a um laço que possui uma ou mais condições de parada, e dentro desse laço existem funções ou condicionais que verificam o estado do jogo (comeu todas as bolas? Foi atingido por um fantasma? Estava energizado?), determinam as ações que devem ser executadas pelos objetos/inimigos (no caso do Pacman, o movimento tanto dele como dos fantasmas) e realizam o redesenho do mapa com o resultado das ações do jogador. Em jogos como xadrez, a partida não avança enquanto o jogador não entrar com algum comando. No Pacman é diferente: se o jogador ficar parado sem fazer nada, os fantasmas continuam andando pelo mapa, caçando vorazmente o jovem comilão amarelo. Nesse caso, o laço do jogo possui um ***tick***, como se fosse um ponteiro do relógio, e a cada vez que avança, as ações do laço do jogo são executadas. Isso permite que os jogos fiquem dinâmicos e as

inteligências artificiais bem-implementadas (ou roubando! Alguns desenvolvedores deixam a inteligência saber mais coisas do que deveria para deixar o jogo mais difícil para você!), tornando a nossa experiência mais emocionante!

Outra parte importante na programação de jogos são as noções de **captura de comandos** e **detecção de eventos**: caso o jogador aperte para cima, o Pacman deve começar a se mover naquela direção, a menos que tenha uma parede. E se ele passar por uma bolinha? Então ela será apagada do mapa. E se bater em um fantasma? Aí depende, se estava em condições normais, o Pacman perde uma tentativa e volta para o ponto de início do mapa, se estava energizado, ele derrota o fantasma e ganha pontos. Perceba que tanto o comando para cima como a colisão com o fantasma (quando o Pacman bate nele) e com a bolinha (quando o Pacman passa por cima dela) são eventos que mudam o estado do jogo. O programa deve ser capaz de, a cada ciclo de interação, detectar quais desses eventos ocorreram e, validando com o estado atual do jogo, realizar as alterações necessárias para apresentar o novo estado do jogo. Esse processo passa por validações, inclusive de qual evento foi acionado primeiro, já que podem existir eventos conflitantes. Imagine um jogo online de combate entre jogadores, como o *League of Legends*: a sequência em que os feitiços e habilidades dos jogadores são acionados pode ser essencial para o resultado da partida, e uma ordem de avaliação equivocada dos eventos gerará um resultado inconsistente (leia-se: catastrófico e com muito xingamento) para o jogo!

Figura 06 - O Pacman quase conseguiu se salvar pegando a bolinha, mas o evento de colisão com o fantasma aconteceu primeiro!



Fonte: http://www.mobygames.com/game/amiga/pacman/screenshots/gameShotId,715203/. Acesso em: 26 ago. 2016.

Um detalhe da captura de comandos: se você quer o seu jogo multiplataforma, então provavelmente precisará codificar funções diferentes levando em conta cada dispositivo possível de interface do jogador. Então normalmente se faz uma função para capturar o *touch* do *smartphone*, uma para mouse/teclado, uma para controles... É um retrabalho necessário para alcançar as diferenças entre os dispositivos e/ou plataformas que podem ser executados pelo seu jogo!

Quase tudo o que falamos até aqui corresponde ao que chamamos de **lógica do jogo**, ou seja, são partes da programação preocupadas em como o jogo funciona no sentido da interação com o jogador. Seria a parte da programação totalmente independente do hardware ou sistema no qual ele roda, seja no celular ou no computador, a lógica do jogo deve ser a mesma. Só que existe também toda uma parte de programação voltada às questões de como o jogo vai rodar no sentido operacional da coisa: como o sistema operacional e o hardware das plataformas

serão utilizados para fazer com que o jogo rode suavemente, sem atrapalhar a experiência do jogador naqueles 60 fps (frames por segundo) que todo jogador almeja ter!

Aqui estamos falando de questões técnicas de computação ao que chamamos de parte **operacional** no começo do texto: como o jogo será gerenciado em termos de memória. No Pacman, todo o mapa é visível na tela, mas e se não fosse? Eu colocaria o mapa todo na memória, arriscando deixá-la cheia e com isso perder desempenho de execução? Ou eu carregaria só a parte visível e arriscaria a possibilidade de falhas no desenho da tela porque o hardware ainda não disponibilizava a próxima parte do mapa carregado na memória para mostrar? Essas questões de como será utilizada a memória, o processador, armazenamento físico em HD, comunicação em rede e outros detalhes técnicos da execução do programa compõem talvez uma das partes mais difíceis da programação de um jogo. E sabe por quê? Porque cada plataforma tem seus padrões e formas de acessar/controlar esses dados, e o programador teria de fazer uma codificação personalizada para cada plataforma. Assim, caso deseje fazer um jogo multiplataforma, prepare-se para codificar várias funções que fazem coisas similares!

Acho que esses poucos exemplos já nos passam uma ideia de como um jogo é feito e do trabalho necessário para gerenciar tudo isso, não é? E estamos falando de *Pacman*, um jogo que em termos de programação é simples de fazer. Imagina agora esses jogos mais novos, como *No Man's Sky*, que possui um universo gerado de forma aleatória e com trilhões de lugares para serem explorados, ou o novo *Doom* com seus gráficos e animações ricos em detalhes... E nem falamos sobre o trabalho que dá desenhar na tela esses jogos 3D!

Figura 07 - Será que é fácil gerar todo esse planeta da imagem aí?

Fonte: http://www.idigitaltimes.com/no-mans-sky-cruelly-reveals-gorgeous-new-screenshot-539195. Acesso em: 26 ago. 2016.

Calma! Nem tudo está perdido! Após alguns anos desenvolvendo jogos, as empresas perceberam que havia algo em comum em seus projetos: elas passavam muito tempo codificando coisas similares ao que já havia sido feito em projetos passados. Ora, existem coisas que todos os jogos devem ter, não é mesmo? Todo jogo precisa controlar um estado de informações; desenhar na tela para o jogador poder ver o que está acontecendo; capturar os comandos do jogador para saber qual decisão foi tomada; e por aí vai. Eles perceberam que se fizessem um programa abrangendo a maior parte desses detalhes técnicos que se repetem entre jogos, permitindo que focassem a produção nos detalhes da lógica do jogo e na implementação apenas de funcionalidades peculiares de cada jogo, eles conseguiriam fazê-lo mais rapidamente, reutilizando funcionalidades que já foram testadas várias vezes e, por isso, apresentavam um menor risco da ocorrência de erros. E assim surgiram os primeiros motores de jogos!

2 - Motores de Jogos

E então, já captaram o que é um motor de jogo?

Não se aperreiem! O final da última seção já nos deixou algumas pistas:

- Motor de jogo é um tipo de programa;
- Foi criado para facilitar a vida de quem faz jogo.

Bom, dizer que o motor de um jogo é um simples programa seria simplificar demais. Na verdade, podemos dizer que o motor representa um *framework* de desenvolvimento de jogos, já que ele possui várias funcionalidades previamente implementadas em sua estrutura e, assim, o desenvolvedor precisa apenas "completar" e alterar o código para criar o seu jogo dentro da estrutura apresentada. Os motores costumam apresentar várias funcionalidades mais gerais, como:

- Motor gráfico, o qual diz como desenhar as coisas na tela (2D e/ou 3D);
- Motor de física, que calcula e determina como os objetos se movem/comportam dentro do jogo;
- Sistemas de suporte a áudio e animações;
- Comunicação de dados via rede;
- Gerência de memória;
- Gerência de componentes do jogo e fluxo de execução;
- Ferramentas de scripting para permitir customização de alguns componentes do motor;
- Inteligência Artificial.

Figura 08 - Um motor como o Unreal 4 tem várias ferramentas, desde edições simples de modelos 3D, até criação de animações e programação de eventos. Isso permite o trabalho de pessoas com diferentes níveis de conhecimento técnico!

Fonte: https://www.unrealengine.com/what-is-unreal-engine-4. Acesso em: 26 ago. 2016.

Não se preocupem, falaremos um pouquinho mais sobre essas funcionalidades na próxima aula. O importante para absorverem agora é que o motor já possui uma base de implementação pronta para você usar e estender com código próprio. Dependendo do motor, isso pode ser feito de forma mais rudimentar, através do uso de bibliotecas de código ou, de forma mais elaborada, com uma interface contendo programas para edição do código, edição de modelos 2D/3D, gerenciamento de recursos, edição de animação, etc.

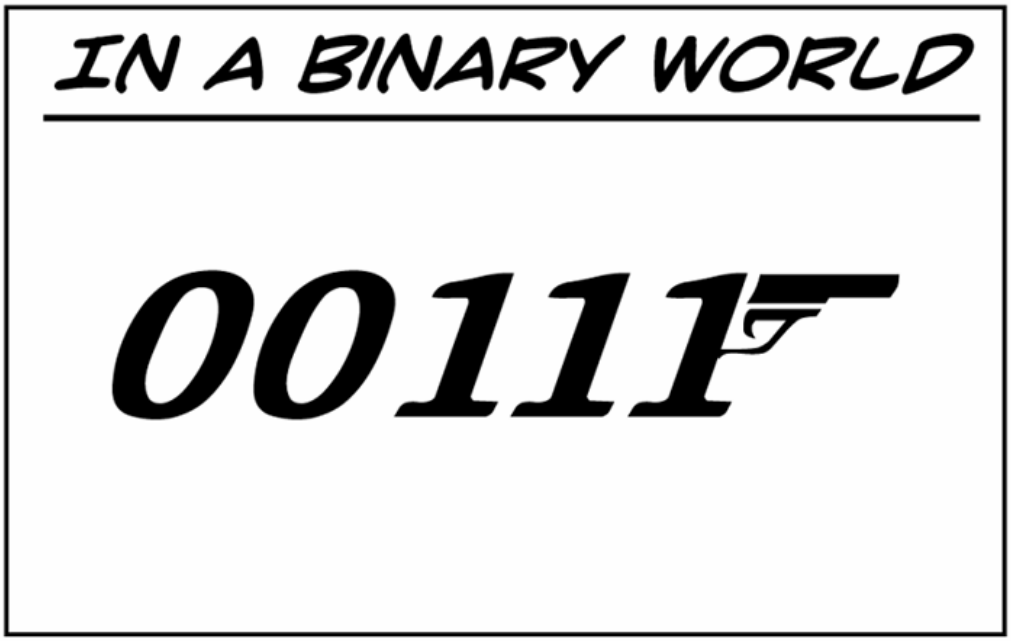
Um detalhe importante é que o motor não substitui todas as ferramentas necessárias na produção de um jogo. Embora alguns até possuam uma capacidade de criar modelos 3D básicos e animá-los, dificilmente conseguirão substituir uma ferramenta específica para esse fim, como o Blender ou Maya (ferramentas de modelagem 3D). Algo que eles fazem é permitir que o desenvolvedor carregue recursos produzidos nessas ferramentas de forma intuitiva e com um bom suporte para diferentes formatos. Daí entramos no segundo tópico: facilitar o desenvolvimento.

O motor de jogos pode ser visto como uma camada intermediária, ou *middleware*, entre a codificação do jogo e o hardware no qual ele é desenvolvido ou executado. Fazendo uma analogia com a programação: o seu computador só entende 0 e 1, a famosa linguagem binária de máquina, correto? Agora imagine o trabalho se você tivesse que programar dessa forma! Faça um programa que leia dois números e apresente a soma:

```
0010101110101
010101011010
111011100101
110101010101
```

Impossível, não é? Embora seja essa a maneira como o computador entende o código, é muito difícil para nós decorarmos como escrever cada instrução e função de 0 e 1. Imagina ler esses números e achar os erros depois! Com certeza, não existiria a programação de computadores como temos hoje. Por isso, foram criadas as linguagens de programação, elas são uma forma em alto nível para que possamos descrever programas em um formato mais próximo da nossa linguagem natural (no caso, inglês), e deixamos a cargo do compilador/interpretador da nossa máquina a árdua tarefa de transformar o nosso programa em código binário para a máquina executar.

Figura 09 - Em binário, até os filmes de James Bond seriam mais difíceis de entender!



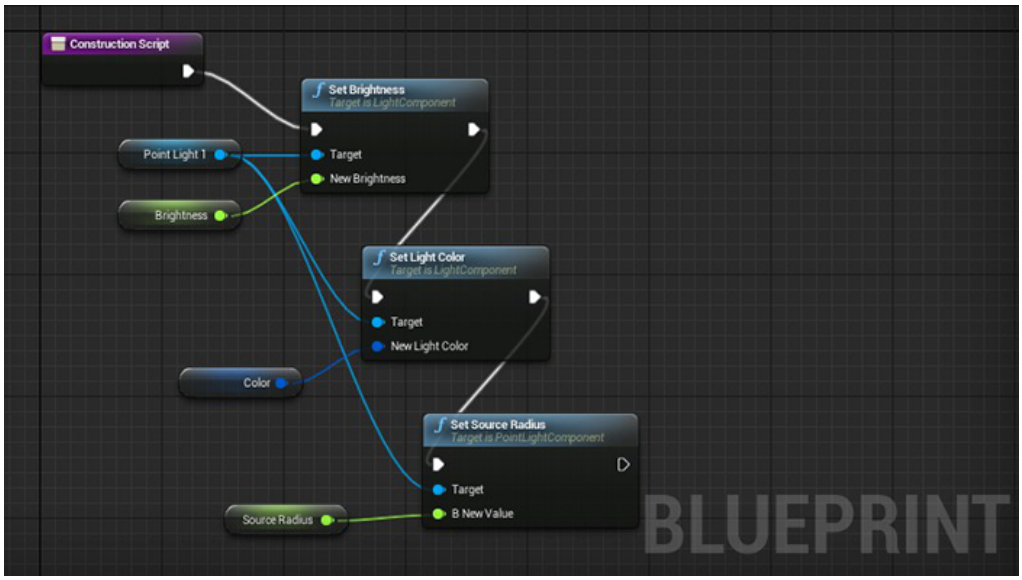
Fonte: http://iiafw.com/tag/binary-world/. Acesso em: 26 ago. 2016.

Analisando o motor de jogo por essa ótica, o que temos é algo similar, como implementar todo o controle de processos, leitura de arquivos, gerência de memória, controle de entrada e saída, dentre outros aspectos da máquina. Isso seria uma grande dificuldade para nós, o que tornaria a tarefa de se criar um jogo uma tarefa digna dos desafios de Hércules! Então o motor dá uma de compilador: ele deixa a nosso cargo a implementação da lógica do jogo, e a maior parte do "trabalho pesado" de interação com o sistema ele faz de modo transparente para o usuário (ou seja, nem vemos!). E como são tarefas que existem em basicamente todo jogo, o motor permite que reutilizemos essas funcionalidades prontas, e isso nos traz duas grandes vantagens:

- Não precisamos fazer novamente;
- Essas funcionalidades já foram testadas e validadas por várias outras pessoas.

Esse reuso permite não apenas a aceleração do projeto atual, como também a execução de vários projetos utilizando o mesmo motor (claro, se a empresa tiver gente suficiente para isso). Por ter componentes próprios para tratar das questões gráficas, de som e de animação, os motores de jogos costumam ser usados também para outras aplicações, como a produção de animações. Inclusive, os motores tentam prover uma interface em que se consegue editar e construir boa parte do jogo sem necessariamente programar, com ferramentas visuais e intuitivas que permitem aos profissionais de outras áreas, como artes e design, interagirem mais facilmente com o motor.

Figura 10 - Muitos motores, como o Unreal, possuem uma linguagem gráfica que facilita a implementação da lógica do jogo, sem precisar digitar uma linha de código!



Fonte: http://blog.digitaltutors.com/unreal-engine-4-vs-unity-game-engine-best/. Acesso em: 26 ago. 2016.

O motor fica responsável por vários aspectos básicos de um jogo. Por exemplo, a inicialização/finalização do jogo ocorre por meio de procedimentos já implementados no motor. Isso é importante porque, ao iniciar o jogo, um conjunto de informações deve ser alocado e estruturado na memória, e quando a aplicação terminar, é necessário "limpar" essa informação para não deixar "lixo" na memória e prejudicar a performance da máquina. Essa gestão também é importante durante a execução do jogo: imagine que seu jogo ocupe toda a memória do seu computador, devido a uma má gestão feita pelo programador. Para compensar essa falta de memória, o sistema operacional começará a trabalhar com memória virtual, usando parte da memória do disco para armazenar dados dos programas em execução, como um novo nível de hierarquia de memória (não lembra bem disso? Dá um pulo na aula sobre memórias da disciplina de Arquitetura de Computadores do básico. Ou olha na internet para refrescar a memória!). O problema é que o acesso ao disco é muito mais lento do que o acesso a RAM, causando uma queda significativa na performance do jogo. Para evitar que isso aconteça, é essencial manter a memória da maneira mais eficiente possível, removendo elementos que não estão sendo mais usados e minimizando o desperdício ao executar a aplicação. Essa gerência manual é muito trabalhosa, e o motor cuida de boa parte dela!

O motor também é responsável pela execução ordenada do jogo. Normalmente, os jogos possuem várias fases ou partes, que são interligadas em um mapa, e o jogador pode navegar livremente por elas (ou não!). Ao construir o jogo, o desenvolvedor indica no motor a sequência ou ligações entre os espaços do jogo e, a partir dessa sequência determinada, o motor irá encadear a carga desses espaços do mundo do jogo na ordem correta. Além disso, o motor implementa o ciclo do jogo de modo que o desenvolvedor precisa apenas especificar os eventos que ocorrerão a cada iteração do ciclo.

Empresas maiores costumam desenvolver motores próprios ou customizar um motor existente com a adição de novas funcionalidades, através de *plugins*. O *plugin* é um programa que é adicionado ao motor para que a nova funcionalidade seja acessível dentro da estrutura utilizada no desenvolvimento do jogo. Dentre os principais motores disponíveis no mercado, temos:

- Construct 2
- Unity
- Unreal 4
- Cry Engine
- GameMaker Studio
- OGRE 3D
- Torque

Figura 11 - Alguns dos principais motores do mercado.

Fonte: http://www.pixelprospector.com/the-big-list-of-game-making-tools/. Acesso em: 26 ago. 2016.

Durante o curso, usaremos *engines* com o propósito geral. Esses motores possibilitam a criação de jogos tanto 2D como 3D, e possuem muitos recursos prontos, normalmente disponibilizados através de uma loja ou *asset store*. Dentre as opções existentes, o Unity e o Unreal 4 se destacam por disponibilizarem versões gratuitas para desenvolvimento.

Curioso para conhecer os componentes do motor em maiores detalhes? Então vamos correndo para a próxima aula! Até lá!

Pontos-Chave

Nesta aula, começamos a conhecer o motor de jogos, uma ferramenta essencial no desenvolvimento de qualquer jogo. Vimos também outros elementos:

- Jogos são programas que exigem um padrão de codificação complexo.
- Um jogo normalmente pode ser dividido em parte operacional, lógica do jogo e visualização.
- O estado do jogo se refere às informações que definem o andamento da partida.
- O ciclo do jogo corresponde à sequência contínua de ações, a saber: verificação de comandos/eventos, validação e alteração do estado do jogo, atualização das informações para o jogador.
- Cada plataforma/sistema possui características únicas que podem levar a partes de códigos específicas para cada uma delas.
- O motor de jogos é uma ferramenta que facilita o desenvolvimento de jogos, abstraindo a parte operacional da programação e permitindo o foco na lógica do jogo.
- Os motores possuem vários componentes prontos, como motor gráfico, motor de física, captura e mapeamento de comandos, áudio e animações, etc.
- Um motor provê uma abstração de alto nível para o desenvolvedor, permitindo que pessoas com menos conhecimento técnico utilizem a ferramenta para produção de jogos e animações.

Leitura Complementar

Para a leitura complementar dessa semana, alguns vídeos sobre motores de jogos!

- Explicação geral sobre como funciona um motor de jogos:
  https://www.youtube.com/watch?v=DKrdLKetBZE
  https://www.youtube.com/watch?v=2tZK75R2K2c

- Vídeos sobre principais motores gratuitos:
  Unreal 4
  Unity 5
  Construct 2

Autoavaliação

1. Como um motor de jogos facilita o processo de desenvolvimento?
2. Se você fosse implementar um jogo na mão, quais os principais passos para a programação dele?

Referências

ENGER, Michael. **Game engines**: how do they work?. Disponível em <http://www.giantbomb.com/profile/michaelenger/blog/game-engines-how-do-they-work/101529/>. Acesso em: 20 ago. 2016.

GREGORY, Jason. **Game engine architecture**. CRC Press, 2009.

MCSHAFFRY, Mike. **Game coding complete**. Cengage Learning, 2012.

WIKIPEDIA. **Game engine**. Disponível em <https://en.wikipedia.org/wiki/Game_engine>. Acesso em: 22 ago. 2016.