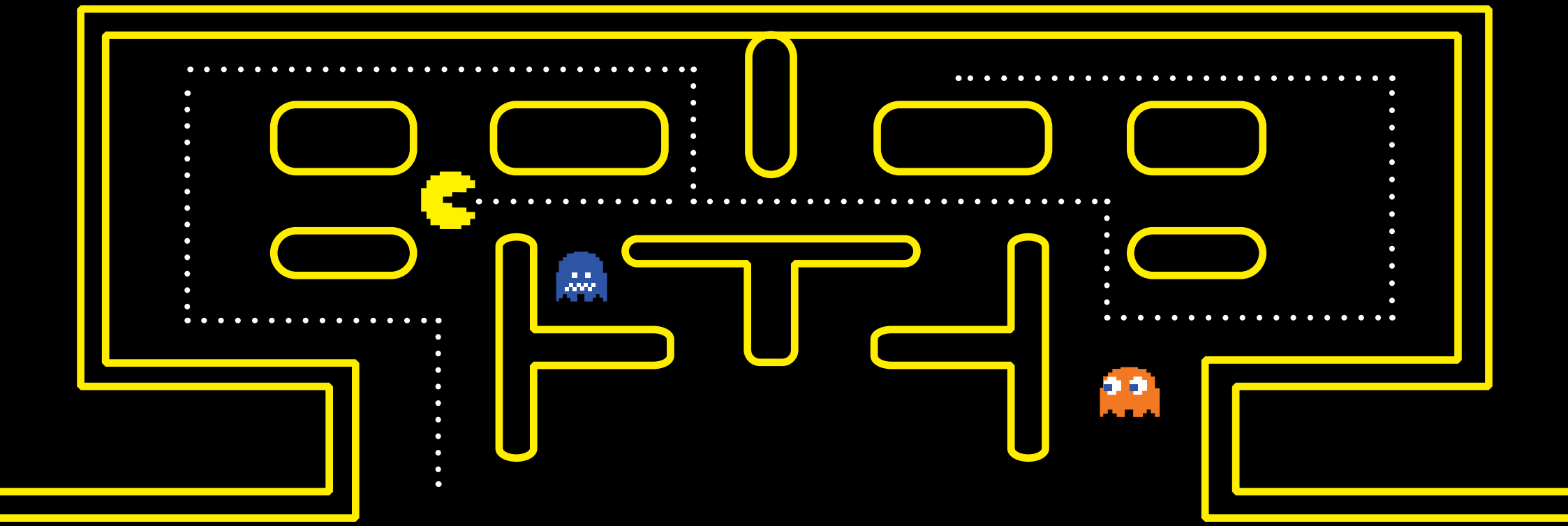
CONTROLE DE VERSÃO

.Técnico em Programação de Jogos Digitais.

Senac



1. Introdução.

Contexto

Iniciamos o estudo de uma prática essencial no desenvolvimento de sistemas de softwares, independentemente de sua natureza. Se você faz parte de uma empresa, certamente trabalhará com uma dessas práticas, assim como se você iniciar um desenvolvimento independente.

O Sistema de Controle de Versão (SCV) serve de suporte ao desenvolvimento, pois permite o controle de modificações nos códigos e nos artefatos (documentos e arquivos) do sistema. Também permite melhor controle para o trabalho paralelo.

Dependendo do tamanho e da complexidade da equipe e do sistema, elabora-se um plano de configuração de *software*. Esse plano contém detalhes de versões de *softwares*, padrões de nomenclatura e identificação dos arquivos, bem como detalhes do ciclo de desenvolvimento pelo qual passa um código, ou um plano, até ser aprovado.

Esta prática é recomendada por padrões internacionais e nacionais de desenvolvimento de sistemas, como o Capability Maturity Model Integration (CMMI) e o MPS.BR. Existem empresas brasileiras que certificam o processo de desenvolvimento e a configuração de *software*. A SOFTEX, por exemplo, é responsável pelo credenciamento e pela certificação de empresas quanto ao processo de *software*.

Mas por que utilizar o controle de versão? Quando estamos desenvolvendo um jogo, por exemplo, trabalhamos com muitos arquivos e muitas equipes de programação e de arte. Imagine a quantidade de arquivos criados e modificados, sendo que cada modificação não necessariamente estará correta ou adequada. Então, como fazer para, neste caso, não se perder ao retornar um arquivo para um estado anterior?

E se dois programadores estiverem trabalhando em um mesmo arquivo de código, porém em partes diferentes, como viabilizar a integração entre as diferentes modificações? Nesses casos, o controle de versão entra em ação criando versões, também conhecidas como revisões, de um arquivo e permitindo registrar e acompanhar as modificações realizadas, podendo retornar, facilmente, a uma versão anterior e ainda realizar a integração entre os arquivos de código modificado e original automaticamente.

Mas, então, o que é um sistema de controle de versão? É uma ferramenta de suporte que permite controlar a modificação de arquivos armazenados em um repositório. Os arquivos sob controle, a cada alteração, geram uma revisão, uma "cópia", que permite que sejam monitorados. Este processo pode ser visualizado no esquema ao lado.

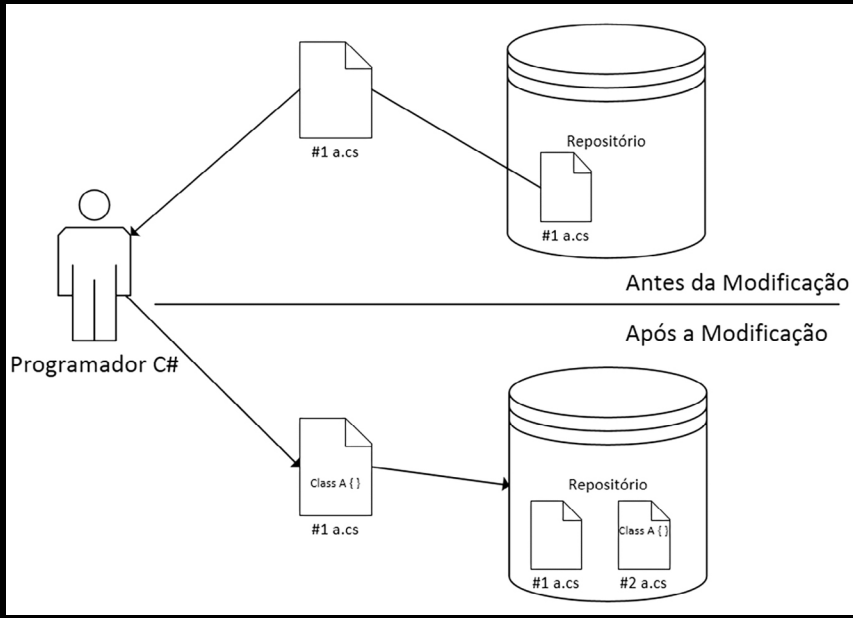
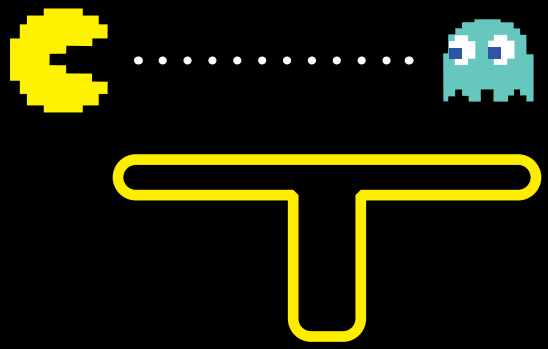


Figura 1 – Representação de um processo de modificação de arquivo controlado por um sistema de controle de versões.



O diagrama mostra um programador pegando um arquivo de código c# com o nome a.cs e versão #1 de um repositório de código, realizando aq inclusão de código, a declaração de uma "classe A{}" e enviando novamente para o sistema de controle de versão, que cria uma nova versão, a #2. Note que o sistema mantém uma "cópia" no repositório sem excluir a versão anterior. Na realidade, esses sistemas mantêm apenas as alterações que foram feitas com relação ao arquivo anterior de forma a economizar espaços de armazenamento, sendo esta outra vantagem de utilização desses sistemas.

No mercado, existem inúmeras ferramentas, comerciais e livres, para realizar o controle de versão. O sistema Subversion é um dos mais

conhecidos. Ele é livre e vastamente utilizado. Para utilizar esse sistema, há a ferramenta Tortoise-SVN, que possui uma interface gráfica para manipulação e controle de modificação dos arquivos.

Outro sistema muito conhecido e vastamente utilizado é o Git, que foi criado para realizar o controle de versão do desenvolvimento do sistema Linux. Este sistema é um dos mais utilizados e possui suporte em vários *frameworks* e IDEs de desenvolvimento de *software* e *games*.

Para utilizá-lo, existem sistemas *web* e *desktop* com interface gráfica para diversas plataformas. As ferramentas que possuem o sistema Git por base são: GitHub e Bitbucket, ambas pagas em função do número de usuários, além da Perforce, entre outras. A GitTree, disponível no site do Git, é livre, e a GitHub também, porém o repositório onde os arquivos são armazenados fica público nesta versão. Algumas empresas possuem seus próprios sistemas de controle de versão, como a IBM, que possui a Rational ClearCase. Esses sistemas podem ser utilizados com comandos em linha, manipulação textual.

As IDEs Visual Studio e NetBeans, como também o motor de jogo Unity 3D, possuem suporte ao sistema Git. Vamos ver a seguir como trabalhar com o Git integrado com a Unity 3D. Empresas como Ubisoft, EA etc. utilizam Perforce. Empresas como Google, nos projetos do Android, Twitter, LinkedIn, Netflix, Facebook etc., utilizam Git.

.2. Prática.

Vamos abordar a prática com o sistema Git e o cliente GitHub Desktop (o site é em inglês). Para isso, seguimos os seguintes passos:

1. Primeiramente, criar uma conta grátis (free acount) no GitHub (<https://github.com/>).

a) Informar os dados de login.

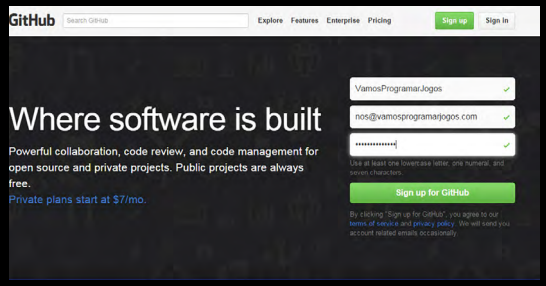


Figura 2 – Tela da página de internet do GitHub para realizar a assinatura (sign up). A imagem apresenta os campos de formulário para informar o nome de usuário, o e-mail e a senha. Após preencher o formulário, o usuário deve clicar em um botão escrito Sign up for GitHub.

b) Escolher o tipo de conta.

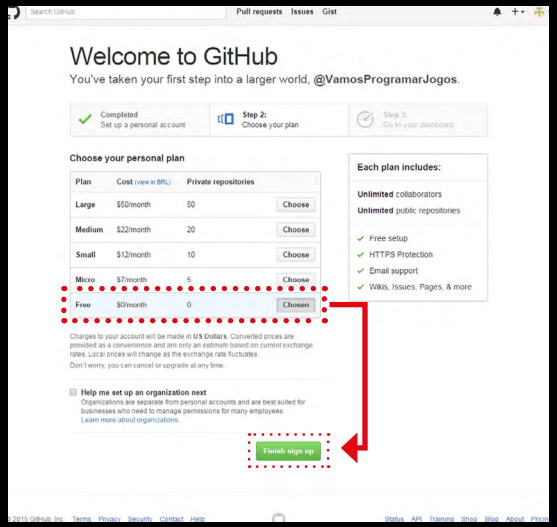


Figura 3 – A imagem apresenta as opções de plano de conta. Está indicado com um retângulo vermelho o tipo de plano gratuito (Free). Após selecionar o plano, para finalizar a assinatura, o usuário deve clicar no botão com o seguinte texto: Finish sign up.

2. Criar um repositório novo (**New repository**) com o nome “primeiroJogoAgora”, por exemplo.

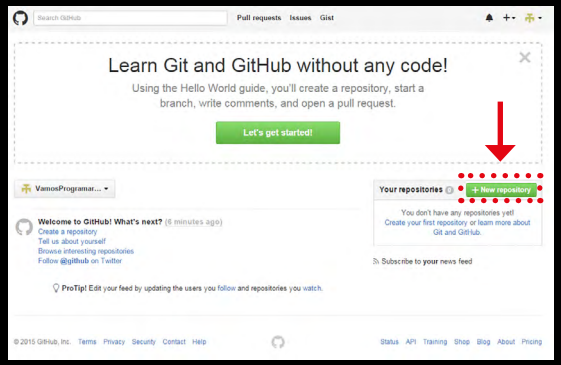


Figura 4 – A imagem apresenta a tela inicial do GitHub após se *logar* na conta pela primeira vez. Está destacado na imagem o botão para criar um novo repositório, com o texto em inglês + New Repository.

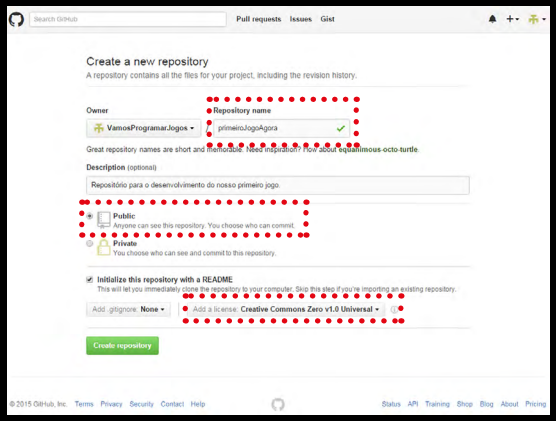


Figura 5 – A imagem apresenta a configuração do novo repositório sendo criado. Deve-se preencher o nome do repositório, no primeiro campo indicado com um retângulo, a descrição do repositório, se ele será público ou privado (lembrando que a conta gratuita suporta apenas repositórios públicos), a seleção do tipo de licença que regulamentará as cópias dos arquivos. Para concluir a criação do repositório, o usuário deve clicar no botão (verde) escrito Create repository.

3. Realizar o download do cliente Git, o aplicativo GitHub Desktop (<https://desktop.github.com/>).

4. Instalar o sistema de controle de versão GitHub, gratuito para pessoa física.

5. Configurar sua conta do GitHub no aplicativo desktop.

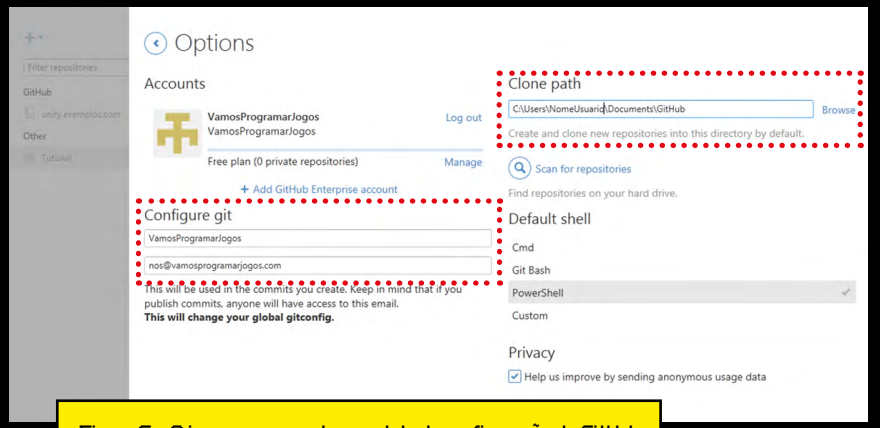


Figura 6 – A imagem apresenta uma tela de configuração do GitHub Desktop. Nesta tela, o usuário tem que definir o diretório onde o repositório será criado no computador. Devem ser configuradas as informações de nome do usuário Git e o e-mail.

6. Volte ao Github *on-line* e acesse seu repositório, após, clique em Clone in Desktop.

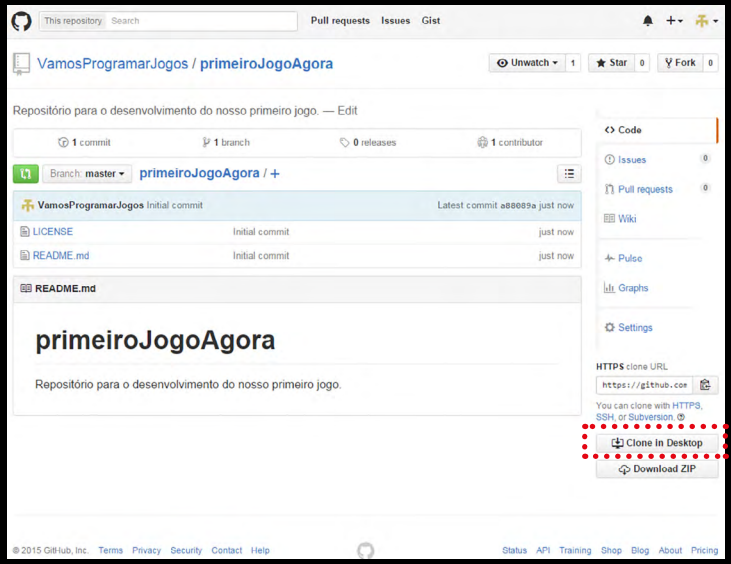


Figura 7 – A imagem ao lado mostra como clonar o repositório *on-line* no computador, integrando o GitHub Desktop com o GitHub *on-line*. Na tela principal do repositório *on-line* do GitHub, tem um botão no menu direito, na parte inferior, que está destacado com um retângulo vermelho com o seguinte texto: Clone in Desktop.

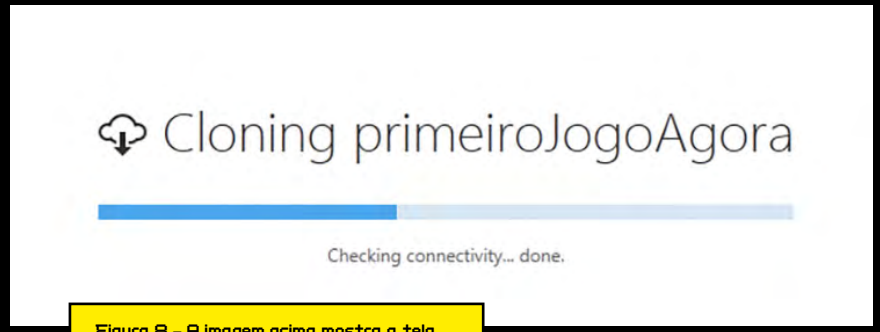


Figura 8 – A imagem acima mostra a tela de progresso da clonagem do repositório on-line no GitHub Desktop.

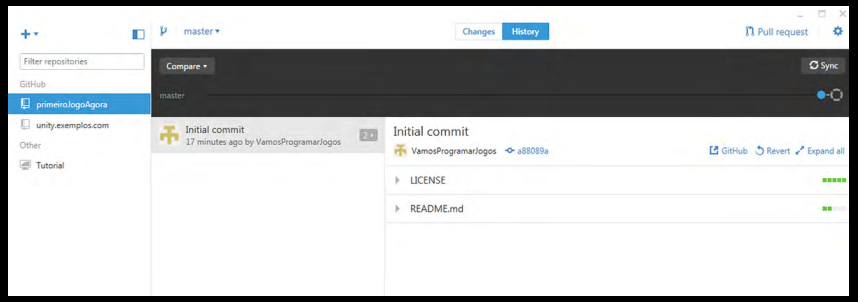


Figura 9 – A imagem mostra a tela principal do GitHub Desktop. Nela, pode ser visto o repositório selecionado: primeiroJogoAgora. Na parte contornada em amarelo, temos o histórico de mudanças do ramo (branch) master (ramo principal), também conhecido como trunk ou main line. Abaixo da linha de histórico, a janela está dividida em duas. Na parte esquerda, temos a lista de commit (as mudanças que foram enviadas para o repositório) e na parte direita, os itens que foram modificados em cada commit.

7. Acessar o diretório onde ficarão os arquivos do projeto do jogo: C:\\usuarios\[seuUsuário]\Meus Documentos\GitHub.

8. Criar um projeto novo na Unity e salvá-lo no diretório do GitHub.

Pronto! Está feita a integração da Unity com o sistema de controle de versão git. Mas o que acontece agora? Ao modificarmos nosso projeto, o sistema GitHub Desktop detecta os arquivos que foram modificados e os destaca para que os enviemos para o repositório on-line.

Pesquise por Git e Github para iniciantes na internet e veja tutoriais em vídeo e páginas com dicas.

Como enviamos os arquivos modificados para o repositório? Selecionamos os arquivos modificados e executamos o comando *commit*. Para exemplificar, foi modificado o arquivo README.md. Incluiu-se uma linha com o texto "PrimeiraModificação". O GitHub Desktop detecta a mudança e nos indica para fazer o *commit* dela para o repositório. Ao "commitarmos" a mudança, informamos do que se trata, preenchendo os campos de formulário com um título e uma descrição da modificação. Após, o usuário deve clicar em **Commit to master** para enviar ao repositório no computador. Para enviar para o repositório on-line, o usuário deve clicar em **Sync.**

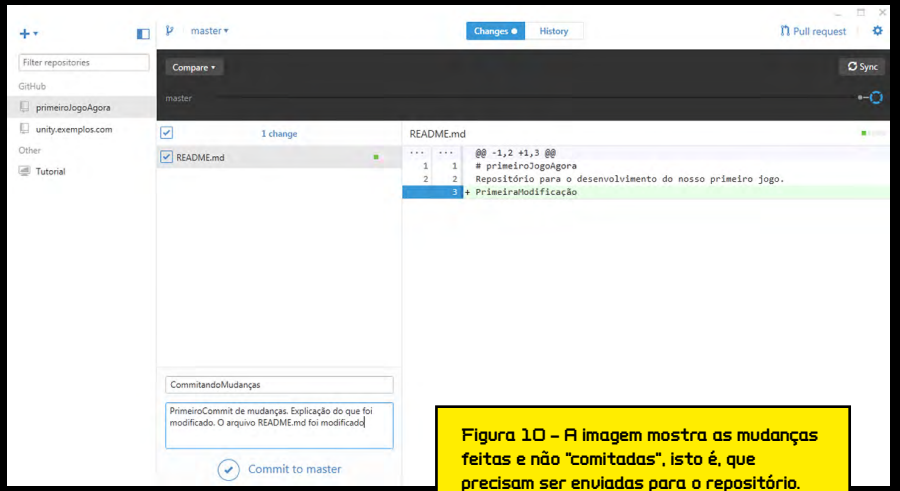


Figura 10 – A imagem mostra as mudanças feitas e não "comitadas", isto é, que precisam ser enviadas para o repositório. Na tela principal, na divisão da esquerda, são mostrados os itens que precisam ser "comitados", na divisão da direita, é mostrado o conteúdo que foi modificado de cada item.

Figura 11 – Tela do menu Version Control.

Esses comandos todos podem ser feitos via linha de comando em um *power shell*. Esse é um processo básico para o controle de versão usando Git. Existem padrões e formas de configurar este processo que facilitam o acompanhamento de mudanças e, caso algo saia errado, facilita o retorno para o arquivo que está estável.

Ainda há a possiblidade de integrar um projeto da Unity utilizando o MonoDevelop, direto com o repositório GitHub *on-line*, eliminando a necessidade de instalar o GitHub Desktop.

No menu **Version Control**, do editor de código da Unity 3D MonoDevelop, destacado na imagem acima, realiza-se a configuração de integração com o repositório GitHub on-line, selecionando a opção **Checkout**. Será exibida uma tela com o formulário para que o usuário insira a URL ou o endereço do repositório, que pode ser obtido na página do GitHub, no local destacado na imagem abaixo.

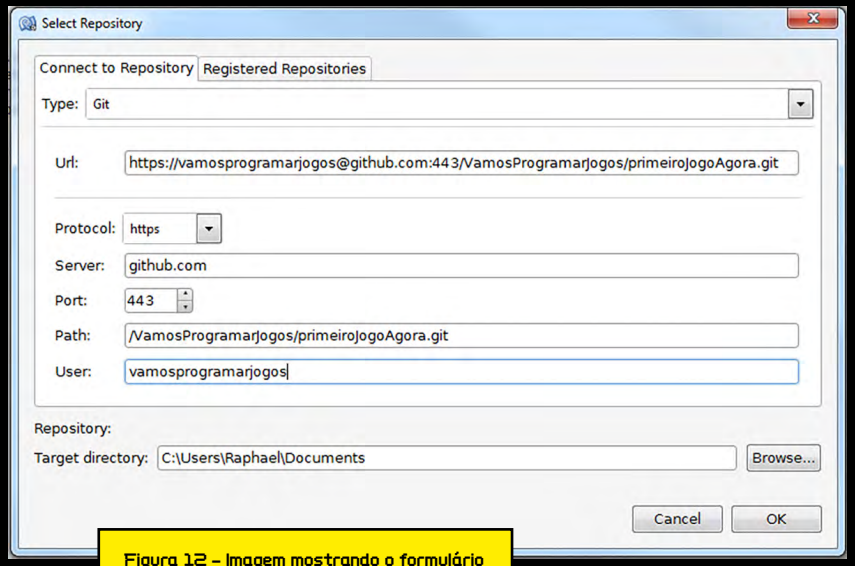


Figura 12 – Imagem mostrando o formulário de seleção de repositório, com dados do repositório criado no GitHub *on-line*.

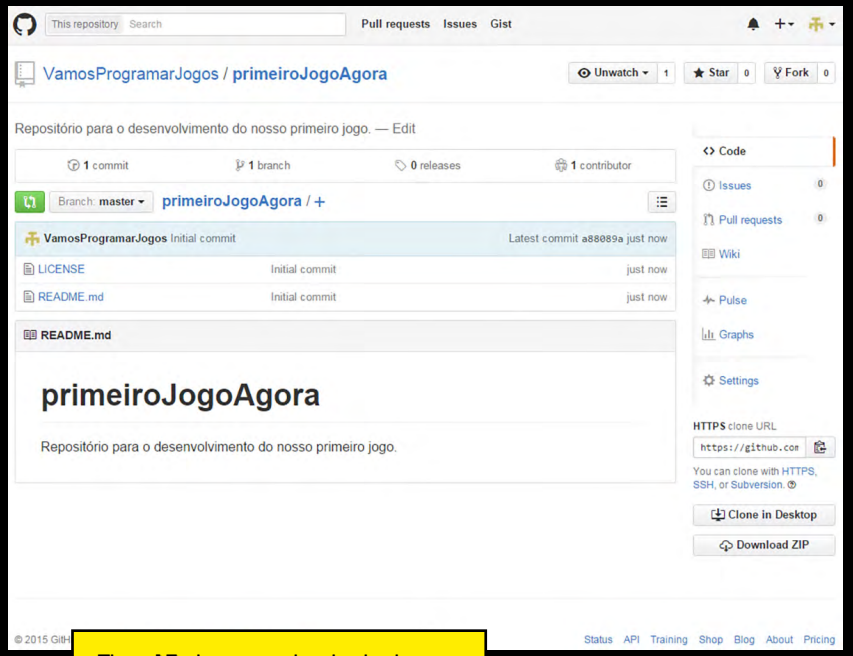


Figura 13 – Imagem mostrando o local para obtenção do endereço do repositório, conforme destacado com retângulo vermelho.

Em seguida, seleciona-se onde serão armazenados os arquivos no computador, o espaço de trabalho. Após realizar uma alteração, basta executar o comando **Commit Solution**, no menu mostrado anteriormente. Para sincronizar com o repositório *on-line*, executa-se o comando **Push Changes**, também no menu citado.'

É importante destacar, ainda, que armazenamos nosso repositório no GitHub *on-line*, porém poderíamos tê-lo armazenado em um servidor de arquivos particular, em algum servidor da empresa, e configurarmos nosso sistema Git para mandar os arquivos (fazer o *commit*) para este repositório. Logo, temos duas possibilidades de configuração: a com repositório armazenado na nuvem (GitHub, por exemplo) e a outra com servidor local. Veja os esquemas ao lado.

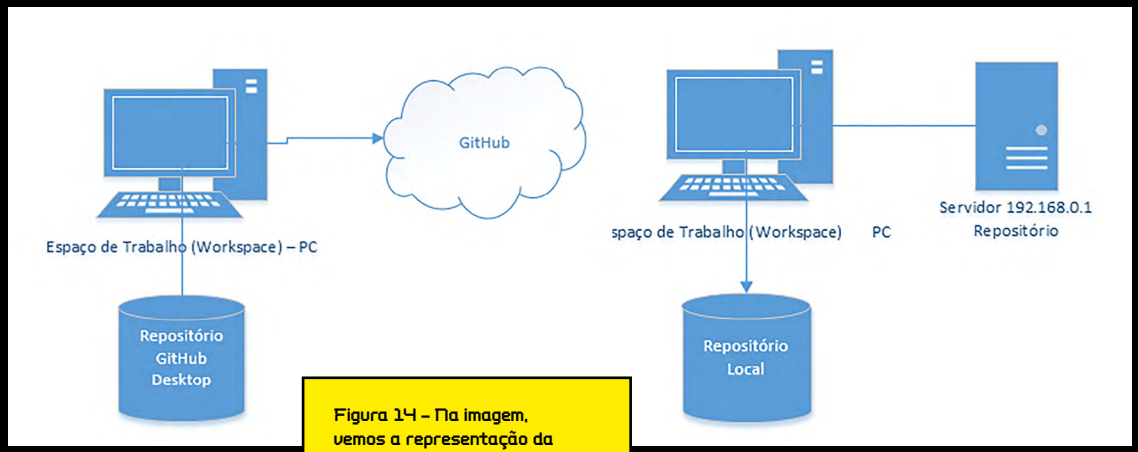


Figura 14 – Na imagem, vemos a representação da conexão de um computador PC, ou um espaço de trabalho, a um repositório local e um repositório na nuvem (esquema à esquerda), ou um repositório configurado em servidor local (esquema à direita).

Por fim, devemos buscar empregar este processo desde o início para que tenhamos o controle do desenvolvimento do jogo. Neste material, nós vimos a importância do controle, as normativas existentes que demandam o controle de versões para dar suporte ao desenvolvimento. Vimos também como criar um repositório no GitHub *on-line* e integrá-lo via GitHub Desktop e MonoDevelop. Destacamos que existem outras formas de configurar o SCV, como, por exemplo, em um servidor local ou na rede da empresa. Vimos os conceitos de repositório e de *commit*. Posteriormente, veremos os padrões de controle de modificações, a criação de ramificações (*branches*) e o uso das tags.

Atlassian Bitbucket. Disponível em: <https://bitbucket.org/>. Acesso em: 26 nov. 2015.

CHACON, Scott; STRAUB, Ben. **Pro Git**. Apress, 2014. Disponível em: <https://git-scm.com/book/en/v2>. Acesso em 13 nov. 2015.

CMMI Institute. Disponível em: <cmmiinstitute.com/>. Acesso em: 26 nov. 2015.

GitHub. Disponível em: <https://github.com/>. Acesso em: 26 nov. 2015.

Git-SCM. Disponível em: <https://git-scm.com/>. Acesso em: 26 nov. 2015.

Perforce. Disponível em: <https://www.perforce.com/>. Acesso em: 26 nov. 2015.

RABIN, Steve. **Introdução ao desenvolvimento de games:** v. 2: Programação. São Paulo: Cengage Learning, 2013.

SOFTEX. Disponível em: <http://www.softex.br/>. Acesso em: 26 nov. 2015.

SOFTEX. **Guia de Implementação** – Parte 2: Fundamentação para Implementação do Nível F do MR-MPS-SW:2012. Disponível em: <http://www.softex.br/wp-content/uploads/2013/07/MPS.BR_Guia_de_Implementacao_Parte_2_20131.pdf>. Acesso em: 26 nov. 2015.

Tortoise SVN. Disponível em: <http://tortoisesvn.net/>. Acesso em: 26 nov. 2015.


Senac