Unity project guidelines

> Esse documento visa tem pro objetivo definir um conjunto de práticas no desenvolvimento de games utilizando Unity e na forma de organizar o fluxo de trabalho para garantir o sucesso no projeto.

Sumário

Repositório de código (versionamento)

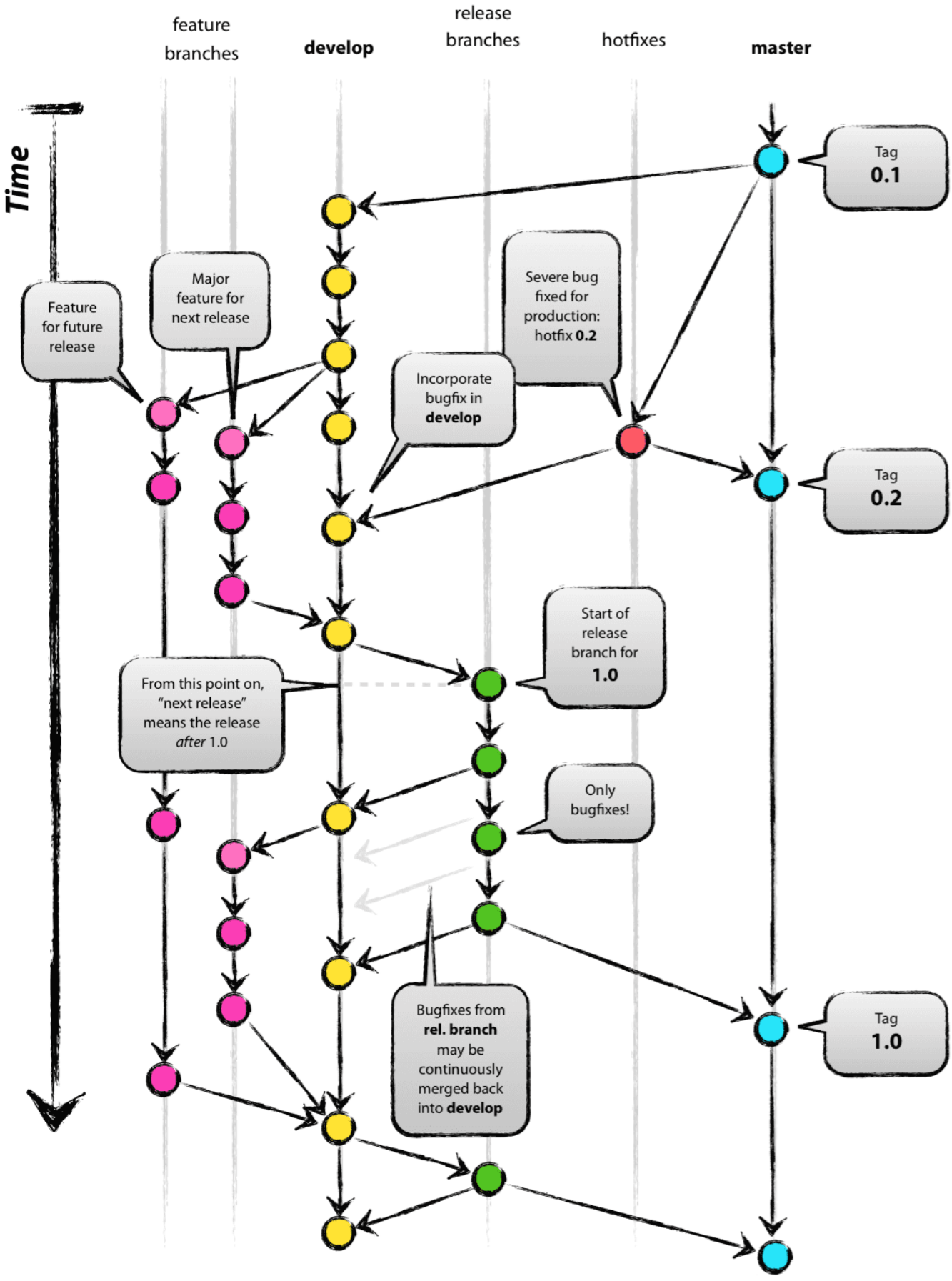
Gitignore

O arquivo de `.gitignore` garante que apenas os arquivos importantes ao projeto sejam persistidos no repositório, além disso evita conflitos em arquivos gerados, que podem ser ignorados no versionamento.

> Referência: link

Branches

Git Flow é um método de trabalho para organizar o desenvolvimento de várias frentes dentro do projeto, de forma a essas frentes não interferirem umas as outras.



O método Git flow é divido em 6 tipos de branches

- **master(main)**
- **develop**
- **feature branches**
- **bugfix branches**
- **hotfix branches**
- **release branches**

master (main)

- **nomenclatura:** master ou main

É responsável por manter a versão mais estável do projeto, geralmente é a versão do jogo que está disponível para outras pessoas utilizarem.

Cada commit nessa branch deve ser adicionado uma **tag** para indicar a versão do projeto que estamos utilizando. Dessa forma fica simples de identificarmos problemas que estão acontecendo direto com os usuários finais.

develop

- **nomenclatura:** develop

É responsável por manter todo o código desenvolvido até o momento no projeto, é uma versão menos estável do que a master, porém já deve conter código finalizado e não em desenvolvimento.

feature branches

- **nomenclatura:** `feature/<número da task>-<nome da feature>`
    - número da task: é referente ao card que a feature foi requisitada, nesse card estão mais informações sobre a feature
    - nome da feature: nome resumido do que a feature entrega para o projeto
- **resumo do fluxo**
    - Abertura da branch pela develop
    - Desenvolvimento
    - Testes
    - Atualização da branch feature pela develop
    - Resolução de conflitos (se existir)
    - Envio do pull request para code review
    - Adequação da branch baseado no feedback
    - Merge na branch de develop

As branches de features são branches abertas da develop onde serão desenvolvidos todos os recursos necessários para a implementação de um feature no projeto.

Uma feature no projeto é um conjunto de tarefas com escopo fechado que devem agregar alguma coisa ao projeto.

Quando o desenvolvimento de uma feature é finalizado e todos os testes foram executados, o desenvolvedor responsável deve fazer um pull request para a branch de develop.

bugfix branches

- **nomenclatura:** `bugfix/<número da task>-<nome da bugfix>`
    - número da task: é referente ao card da bugfix, nesse card estão mais informações
    - nome da bugfix: nome resumido do que a bugfix entrega para o projeto
- **resumo do fluxo**
    - Abertura da branch pela develop
    - Desenvolvimento
    - Testes
    - Atualização da branch feature pela develop
    - Resolução de conflitos (se existir)
    - Envio do pull request para code review
    - Adequação da branch baseado no feedback
    - Merge na branch develop

As branches de bug fixes são abertas para resolver bugs encontrados na versão atual da branch develop.

hotfix branches

- **nomenclatura:** `hotfix/<número da task>-<nome da hotfix>`
    - número da task: é referente ao card da hotfix, nesse card estão mais informações sobre a hotfix
    - nome da hotfix: nome resumido do que a hotfix entrega para o projeto
- **resumo do fluxo**
    - Abertura da branch pela master
    - Desenvolvimento
    - Testes
    - Atualização da branch feature pela master
    - Resolução de conflitos (se existir)
    - Envio do pull request para code review
    - Adequação da branch baseado no feedback
    - Merge na branch master e criação da tag
    - Merge na branch develop

As braches de hot fixes são abertas para resolver bugs encontrados na versão atual da branch master, ou seja, bugs que estão no projeto utilizado pelos usuários finais.

release branches

- **nomenclatura:** `release/<número da task>-<nome da release>`
    - número da task: é referente ao card que a release, nesse card estão mais informações sobre a release, como a release notes

    - nome da release: nome resumido do que a release entrega para o projeto, como talvez as principal funcionalidade

- **resumo do fluxo**

    - Abertura da branch pela develop
    - Testes
    - Correção de bugs
    - Atualização da branch feature pela master
    - Resolução de conflitos (se existir)
    - Envio do pull request para code review
    - Adequação da branch baseado no feedback
    - Merge na branch master e criação da tag
    - Merge na branch develop

As branches de release são abertas para publicar a versão do projeto que está em desenvolvimento para uma versão estável na branch master.

Nessas branches estamos mais interessados em testar a integração de tudo que foi desenvolvido por todas as frentes.

Tags

As tags são versões do código que são atribuidas um nome fixo dentro do versionamento. As tags são importantes que dessa forma podemos distribuir o nosso projeto para várias pessoas e recolher seus feedbacks tendo como referência a versão do projeto que as pessoas tiveram contato. Por esse motivo é de suma importância que o projeto tenha descrito qual a versão está sendo executada.

Versionamento por número

> [major].[minor].[change]

- **Major:** determina as versões globais do projeto

    - `< 0`: projeto ainda não concluíu o MVP (Minimum viable product)
    - `= 1`: projeto já concluíu o MVP
    - `> 1`: grandes alterações no projeto que impactam no estado atual do projeto, por exemplo, implementação de um novo sistema que irá alterar o fluxo do jogo, adição de um novo modo no jogo.

- **Minor:** determina as adições de funcionalides menores no projeto. Cada nova grande funcionalidade aumentar esse número em 1.

- **Change:** mudanças pequenas no projeto, geralmente correções de bugs e melhorias em cima das versões. Cada nova mudança deve aumentar esse número em 1.

Commits

As mensagens de commits devem ser simples, sucintas e representar apenas o mais importante que foi desenvolvido.

Cada commit deve resolver apenas um problema, dessa forma podemos manter um histórico das atividades que foram desenvolvidas.

- **Formato da mensagem de commit**
  - Iniciar utilizando um emoji para representar o que foi desenvolvido
    - Para os emojis utilizar como referência o site link
  - Uma mensagem referênte ao que foi desenvolvido de até 80 caracteres

Exemplos de mensagem de commit

> ✨ Sistema de movimentação do personagem

> 🐛 Personagem principal ficava preso na animação de cair

> 💄 Nova barra de experiência para o personagem principal

> 🚧 Movimentação de patrulha dos inimigos

> ♻️ Sistema de cálculo de dano baseado em atributos

- ✨: adição de nova funcionalidades ao sistema
- 🐛: correção de bug
- 💄: adição ou atualização e UI ou estilo
- 🚧: funcionalidade ainda em desenvolvimento, muito utilizado para garantir o backup dos arquivos desenvolvidos, ou certo ponto de desenvolvimento, porém a funcionalidade ainda não está totalmente finalizada
- ♻️: refatoração de parte ou total funcionalidade do projeto

Pull requests

Toda integração entre as várias frentes de desenvolvimento seram concatenadas por meio do pull request.

Vantagens de utilizar pull requests

- Propagação de conhecimento na equipe
- Garantir de qualidade e padronização do projeto
- Confiabilidade no trabalho desenvolvido

A criação dos pull requests seguem o seguinte formato:

- **feature** finalizada para **develop**
- **bugfix** finalizada para **develop**
- **hotfix** finalizada para **master**
- **release** finalizada para **master**

Nesses casos o pull request deve ser avaliado por todos os membros competentes ao assunto.

Formato de um pull request

Quando o pull request é criado ele deve seguir o seguinte formato:

```
## Descrição

- [Pode ser removido] O que foi feito?
- [Pode ser removido] Quais as mudanças que a alteração trás no sistema?

- [Pode ser removido] Como foi feito?
    - [Pode ser removido] Alguma técnica que é legal ser compartilhada?
    - [Pode ser removido] Alguma estrutura/biblioteca foi utiliza?

- [Pode ser removido] Por que foi feito?
    - [Pode ser removido] Qual o intuito de fazer essa alteração?
    - [Pode ser removido] Alguma melhoria de performance foi feita?
    - [Pode ser removido] Alguma refatoração de código?
    - [Pode ser removido] Qual o valor que agora podemos trazer para o cliente?

## Testes realizados

- Testes gerais realizados
  - Cenas de demonstrações criadas
  - Cenas de protótipo criadas
  - Quais cenas do projetos foram testadas
- Testes unitários implementados

## Outras informações

- [Pode ser removido] Existe outra proposta para resolver esse problema que seria
melhor?
- [Pode ser removido] Alguma outra informação relevante?
```

O que está escrito como **[Pode ser removido]** está no template apenas como referência de desenvolvimento.

O pull request deve ser o mais sucinto e direto possível, para ajudar aos outros integrantes do tipo identificar com exatidão o que realmente foi implementado.

Hierarquia do projeto

A fim de facilitar a busca de elementos relacionados e de melhorar a visibilidade de cada asset desenvolvido, faz-se necessário utilizar uma hierarquia mais independente focada no que cada elemento significa dentro do projeto e não na sua função. Dessa forma centraliza-se todos os dados referêntes ao elemento do projeto em um único lugar.

O projeto na Unity deve seguir a seguinte hierarquia

```
project
 ├ Assets
 │ ├ Audios
 │ ├ Decorations
 │ ├ Fonts
 │ ├ GameAssets
```

```
│ │ ├ 🗁Managers
│ │ ├ 🗁Elementos do projeto
│ │ └ 🗁etc
│ ├ 🗁Particles
│ ├ 🗁Shaders
│ ├ 🗁Plugins
│ ├ 🗁Resources
│ ├ 🗁Scenes
│ │ ├ 🗁Demo
│ │ ├ 🗁GameBuild
│ │ └ 🗁Prototype
│ └ 🗁UI
├ 🗁Docs
├ 🗁Packages
│ └ 📜manifest.json
├ 📜.gitignore
└ 📜read.me
```

- Assets: pasta raiz do projeto dentro da Unity, apresenta tudo o que tem respeito global no projeto
    - GameAssets: representa todos os elementos que irão constituir uma cena, focado em elementos que apresentam alguma lógica de programação e são mais dinâmicos
    - Decorations: representa os elementos decorativos globais, geralmente são elementos estáticos e que são reaproveitados em várias cenas
    - Particles: representa elementos de partículos globais, que são utilizados por vários elementos do projeto
    - Shaders: representa os shaders globais, que são utilizados por vários elementos do projeto
    - Scenes: representa as cenas criadas dentro da Unity
        - Demo: cenas de demonstração de uma ou mais funcionalidades dentro do sistema, geralmente utilizadas para testar essas funcionalidades
        - GameBuild: cenas que estão no jogo
        - Prototype: cenas que estão sendo desenvolvidas ou responsáveis por testar várias mecânicas do jogo, podem evoluir para se tornar uma GameBuild

Elementos do projeto são quaisquer elementos que serão inclusos no projeto, por exemplo: inimigos, personagem principal, inventário de ítens entre outros. Esses elementos devem ser hierarquizados de forma indenpendente de forma a todos os recursos envolvidos estejam centralizados, dessa forma melhoramos a visibilidade do projeto.

Pegando como exemplo a hierarquia de inimigos, como um **elemento do projeto**, a hierarquia pode ser apresentada da seguinte forma:

```
🗁Elemento do projeto (inimigos)
├ 🗁Animations
├ 🗁Audios
├ 🗁Materials
├ 🗁Models
├ 🗁Prefabs
├ 🗁Scripts
├ 🗁ScritableObjects (As configurações de cada inimigo)
```

```
├ 🗁Shaders
├ 🗁Sprites
├ 🗁Textures
```

Dessa forma, temos todas as informações referentes a inimigos centralizadas e de fácil acesso. Por outro lado os recursos globais (utilizados por vários elementos) estam contidos nas pastas globais.

Em caso de necessidade, se um **elemento do projeto**, se tornar muito grande, é possível dividir esse elemento em outros separando a hierarquia, por exemplo: se dentro do projeto existem vários tipo de inimigos e eles são muito diferentes entre si, pode-se quebrar numa hierarquia de uma pasta Inimigos geral, e pastas de inimigos para cada tipo de inimigos, dessa forma cada contexto terá seus próprios recursos e forma independente.

Nomenclatura

Nomenclatura de elementos dentro do projeto.

Assets dentro do projeto devem seguir **snake_case**.

Padrão de nomenclatura para assets:

Asset type	Name
Animations	`[animation_name]_anim`
Audios	`[audio_name]_sfx`
Materials	`[material_name]_mat`
Prefabs	`[prefab_name]_pf`
Shaders	`[shader_name]_shader`
Scriptable objects	`[object_name]_so`
Sprite	`[script_name]_sprite`
Texture	`[texture_name]_texture`

Code guides

Code guides são padrões adotados no código para garantir agilidade no desenvolvimento. Dessa forma todos os desenvolvedores de um projeto compartilham um mesmo vocabulário facilitando a comunicação e o entendimento do projeto.

Princípios

Formatação

Quantidade de caracters por linha: 99;

Sonar lint

O Sonar é uma ferramenta com uma opção gratuita de análise de código e pode ajudar muito em evitar bugs e em melhorar o repositório de código de forma geral, auxiliando a detecção de **bad smells** no código (problemas que podem ocasionar bugs ou problemas na redigibilidade e manutenabilidade de um software).

Para o desenvolvimento de código para a Unity o Sonar é ncessário alguns ajustes, já que o fluxo padrão do C# não é integralmente funcional no ambiente da Unity.

O arquivo `resources/.editorconfig` apresenta vários ajustes para utilizar com o Visual Studio Community.

Modo de utilização no Visual Studio Community

- Copiar o arquivo `resources/.editorconfig` para a pasta raiz do projeto
- Abrir o Visual Stuido Community

Organização do trabalho

Nessa seção serão as práticas relacionadas a organização das tarefas e das demandas do dia a dia e como organizá-las

Quadro de tarefas

Tarefas

Descrição de uma tarefa

Critérios de aceitação

Backlog

Priorização de backlog

Releases

Release notes