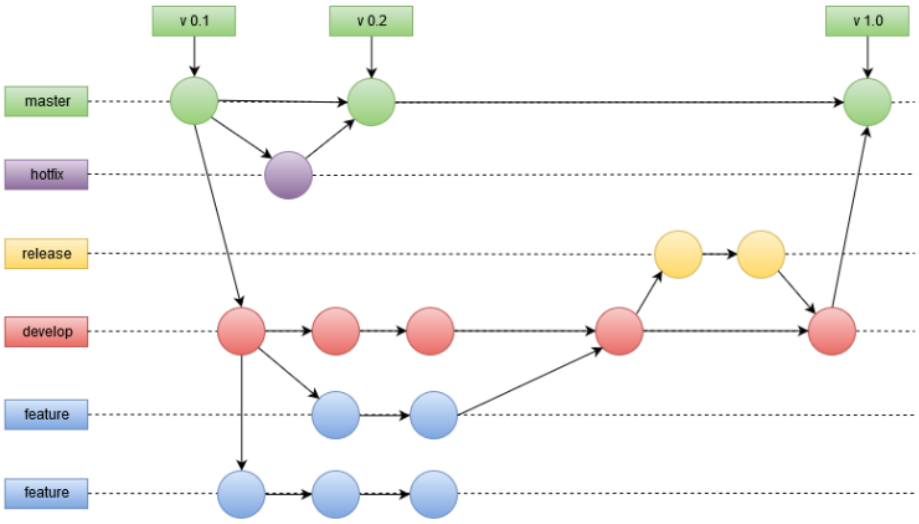
Github:

1. Opis gitflow



Zrzut ekranu 1 Wizualizacja przykładowego GitFlow

W repozytorium znajdują się gałęzie:

- Master

W branchu master umieszczamy wersję ostateczną, czyli released

- Release

jeżeli uznaliśmy, że mamy już wystarczającą ilość nowych funkcji (wszystkie branche **feature** muszą być scalone z **develop** – jeśli nie, to pójdą w następnym wydaniu) tworzymy gałąź **release**. Wychodzi ona z **develop'a** i kończy tym samym dany etap pracy. Utworzenie tej linii oznacza także chęć wydania wersji. W gałęzi **release** dodajemy notatki, aktualizujemy dokumentacje, a także dokonujemy lekkich napraw przed samym wydaniem (to jest też dobry czas na testy integracyjne, QA itp.). Po skończeniu pracy musimy scalić nasz **release** z **master'em** oraz **develop'em**

- Develop

W gałęzi **develop** umieszczamy wszystkie **branche** związane z dodaniem nowej funkcjonalności

- Feature

Jest to gałąź, która ma swój korzeń w **develop** oraz powinna kończyć się w **develop** (w przypadku porzucenia rozwoju funkcjonalności nie scalamy z **develop**). Przechowujemy tutaj wszystkie nowe funkcjonalności oraz elementy, które mają być przetestowane przed wdrożeniem na produkcję.

- Hotfix

Jest to gałąź, która ma swój korzeń w **master**. Wykonywane są w niej zadania, które związane są z naprawą błędów w aktualnie wydanej wersji

Po rozwiązaniu problemu bardzo ważne jest scalenie brancha **hotfix** z branchami **master** oraz **develop**. W **master** umieszczona jest wersja bez błędu (każdorazowe umieszczenie czegoś w gałęzi master oznacza **release**). Należy także scalić poprawkę z **develop'em**, aby móc pracować na najnowszej wersji (czyli tej z poprawką).

Instrukcja git bash

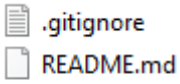
1. Klonujemy repozytorium do siebie na komputer

git clone https://github.com/mapisarek/Survival_Zombie_2D.git

2. Po skopiowaniu przechodzimy do sklonowanego repozytorium

cd Survival_Zombie_2D/

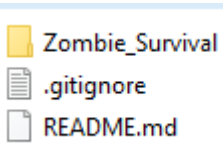
3. W folderze znajduje się teraz zawartość gałęzi master czyli



4. Przechodzimy do gałęzi develop

git checkout develop

5. W folderze pojawia się nam zawartość gałęzi develop



(Folder zawiera projekt w Unity)

6. Żeby korzystać z git flow wpisujemy komende

git flow init

Klikamy enter na wszystkie zapytania

7. Żeby dodać jakąś funkcjonalność do naszego programu wpisujemy

git flow feature start **NAZWA_FUNKCJONALNOŚCI**

Z automatu zostaniemy przełączeni na tą gałąź.

8. Jak wprowadzamy zmiany u siebie na gałęzi, możemy sprawdzić jakie pliki się zmieniają komenda:

git status

9. Aby dodać jakieś pliki do zmian wpisujemy komende

git add … (nazwa pliku) lub git add . (kropka dodaje wszystko)

10. W dodanych plikach rejestrujemy zmiany komendą

git commit -m ‘’informacja o commicie’’

11. Aby wypchać zmiany do developa wpisujemy

git flow feature finish **NAZWA_FUNKCJONALNOŚCI**

Komenda usuwa nam gałąź feature i scala się z gałęzia develop

12. Aby opublikować gałąź ze zmianami na serwerze wpisujemy

git flow feature publish **NAZWA_FUNKCJONALNOŚCI**

Zasady na inżynierie oprogramowania:

1. Commity mają być atomowe czyli robimy commit do każdej operacji którą wykonujemy na projekcie:

- Dodanie klasy
- Dodanie metody
- Ustawienie obiektów
- Wgranie tekstur do funkcjonalności (proponuje nie wgrywać każdej pojedynczej tekstury tylko np. wgranie folderu który będzie np. obsługiwał mape na danym etapie).

2. Commity i nazwy featureów nazywamy po angielsku, commity - według schematu

git commit -m ‘’Task 1: Added menu class” (Numer taska do funkcjonalności bierzemy z trello)

3. Rejestry zmian mają być wprowadzane na bieżąco