Środowisko developerskie - stawianie nowego projektu

Utworzenie środowiska developerskiego dla nowego projektu

Po wykonaniu wszystkich kroków z poniższej instrukcji:

- Będą przygotowane kanały komunikacji dla zespołu tworzącego grę: confluence, JIRA itp.
- Będą utworzone i skonfigurowane repozytoria `git` zawierające przykładowy kod logiki gry oraz klienta gry.
- Będą uruchomione i skonfigurowane lokalne wirtualki, na których będzie odpalona platforma SGS z podpiętą przykładową logiką gry.
- Kod z repozytoriów `git` logiki oraz klienta będzie budowany automatycznie przez taski na serwerze `jenkins.gd`.
- Zespół będzie mógł we własnym gronie prowadzić development i testy gry:
  - Serwer gry uruchamiany na wirtualkach oraz na komputerze developera logiki.
  - Klient kompilowany i uruchamiany na komputerze developera klienta, łączący się z dowolnie wybraną instancją logiki.

Instrukcja nie obejmuje etapu deploymentu gry dla grona odbiorców poza zespołem gry.
Nie obejmuje też tematu robienia integracji z zewnętrznymi sajtami takimi jak Facebook czy GameDesire.
Tym samym wszelkie taski związane z developmentem aplikacji pod kątem integracji, takie jak listy przyjaciół, sharingi, notyfikacje, giftingi, płatności itd. nie są tutaj uwzględnione.
Osobne instrukcje dla poszczególnych integracji znajdują się gdzie indziej.

Jeśli przy wykonaniu któregoś punktu wystąpią problemy, to najlepiej dopisać od razu w tym miejscu komentarz, albo samodzielnie uaktualnić/poprawić instrukcję.

---

1. Ustalenie nazwy kodowej projektu.

- Ustalamy oficjalną techniczną nazwę kodową projektu. Powinna ona jednoznacznie identyfikować nasz produkt, być stosunkowo krótka, unikalna i zawierać wyłącznie małe litery a-z.
  Będzie ona używana w nazwach serwerów, katalogów i plików.
  - Przykłady dobrych nazw: `astropolis`, `lasttemple`.
  - Nazwa nie musi odpowiadać dokładnie naszej oficjalnej nazwie produktu, np. `slots` vs. "SlotIn!".
    Powinniśmy się jednak jej trzymać konsekwentnie w kodzie, aby uniknąć sytuacji, że raz katalog nazywa się `billiards`, a raz `poollivepro`. Utrudniać to może późniejsze integracje.
  - W poniższej instrukcji używamy kodowej nazwy projektu `pandopolis`.
    Jeśli gdzieś napisane jest `Pandopolis`, tzn. że chodzi o nazwę oficjalną produktu - niekoniecznie musi być równa nazwie kodowej.

2. Utworzenie kanałów komunikacji dla projektu wewnątrz zespołu.

Cel:

- Każdy członek zespołu wie gdzie szukać i umieszczać informacje nt. postępów w projekcie.

Czynności do wykonania:

- Zgłaszamy się do administratora `confluence.gd` z prośbą o utworzenie własnego space dla projektu o nazwie "Game - Pandopolis".
  Aktualnie administratorem jest Maciej Mróz .
- Zgłaszamy się do administratora `jira.gd` z prośbą o utworzenie nowego projektu.
  Aktualnie administratorem jest Maciej Mróz.
- Zgłaszamy się do działu OPS z prośbą o utworzenie listy mailowej o nazwie w stylu `pandopolis_team@ganymede.eu` i zapisanie na nią członków naszego teamu.
  Właściwą drogą zgłoszenia jest utworzenie taska w JIRA na stronie http://jira.gd/browse/OPSF.
  Adres taki może być oprócz komunikacji wewnątrz zespołu użyty np. do przesyłania informacji o nieudanych buildach w jenkinsie.
- Zgłaszamy się do działu OPS z prośbą o dostęp do \\STORAGE\graphics dla grafików z naszego teamu. TODO: Ten krok jest potrzebny tylko dla nowych ludzi, imo powinno się go przenieść do procedury zatrudniania pracownika
  Droga zgłoszenia jw.
- Wklejamy na główną stronę space jako wizytówkę projektu zawartość strony Strona nowego projektu - template
  Następnie edytujemy ją umieszczając znane nam informacje dot. naszego produktu.
- Na mail grupowy zespołu wysyłamy linka do ww. strony.

Weryfikacja:

- Pytamy każdego członka zespołu czy otrzymał maila z linkiem do strony wizytówki projektu i się z nią zapoznał. Każdy powinien potwierdzić.

3. Postawienie lokalnych wirtualek.

Cel:

- Wykonane są wszystkie operacje wymagające zależności zewnętrznych. Po tym kroku resztę czynności możemy wykonać we własnym zakresie.

W skład środowiska lokalnego projektu wchodzą 4 wirtualki:

- `db.pandopolis.gd` - serwer bazy danych MySQL
- `solid.pandopolis.gd` - serwer z platformą SGS - obsługa logiki gry
- `chaos.pandopolis.gd` - dodatkowy serwer z platformą SGS - obsługa eksperymentalnych featurów logiki gry
- `web.pandopolis.gd` - serwer, na który trafiają buildy klienta gry i na którym prowadzony jest lokalny development www (integracje z zewnętrznymi sajtami + ew. narzędzia pomocnicze opisane w punkcie 8.)

> Lista oprogramowania, które powinno być na wirtualkach zainstalowane: Wirtualki dla developmentu.

Czynności do wykonania:

- Zgłaszamy się do działu OPS, poprzez JIRA, z prośbą o utworzenie ww. serwerów.
  Należy powołać się na task http://jira.gd/browse/OPSF-3421 i poprosić o utworzenie wirtualek z tych obrazów.
  Task powinien wiązać się z ustawieniem odpowiednich wpisów w lokalnym DNS.
- Upewniamy się że na wirtualkach w pliku `/etc/resolv.conf` ponad nameserverami znajduje się wpis "`search gd`". To jest konieczne aby poprawnie rozwiązywały się nazwy domen `*.gd` z poziomu danej maszyny wirtualnej.
- Zgłaszamy się do działu OPS z prośbą o dodanie namiarów na nowo postawiony serwer bazodanowy `db.pandopolis.gd` do listy serwerów na stronie `pma2.gd`.
  Prosimy równocześnie o instalację certyfikatów do tego phpMyAdmina na komputerach developerów w naszym zespole.
  Przykład takiego taska: OPSF-4454 - Getting issue details... STATUS .

Weryfikacja:

- Logujemy się przez SSH na każdą z wirtualek używając loginu/hasła ze strony-wizytówki w confluence.
  Logowanie powinno się powieść.
- Na wirtualkach `solid` i `chaos` upewniamy się, że mają min. 500MB RAM. Mniejsza ilość RAMu skutkuje później masakrycznie długimi czasami kompilacji logiki gry.
  - Wykonujemy polecenie: `free -o`
  - Jeśli w kolumnie `total` liczba pamięci wskazuje poniżej 500MB, zgłaszamy się do działu OPS z prośbą o zwiększenie jej do min. 512MB.
- Poprzez stronę `pma2.gd` logujemy się do bazy danych MySQL na `db.pandopolis.gd` używając konta `root` i hasła umieszczonego na stronie-wizytówce w confluence.
  Logowanie powinno się powieść.

4. Postawienie działającego serwera gry z przykładową logiką.

Cel:

- Uruchomiona jest przynajmniej jedna instancja platformy SGS służąca do obsługi naszej gry.
  Można się do tego serwera podłączyć używając testera logiki i wykonać przykładowe operacje.

Założenie:

- Mamy działające konto na firmowym gitlabie pod adresem `dev.git.gd`, na które jesteśmy w stanie się zalogować. Jesteśmy w stanie sklonować na swój komputer firmowe repozytoria git.
  Instrukcja zakładania konta znajduje się pod adresem: Internal git server

4.1. Inicjalizacja baz danych.

Czynności do wykonania:

- Poprzez stronę `pma2.gd` logujemy się do bazy danych MySQL na `db.pandopolis.gd` używając konta `root` i hasła umieszczonego na stronie-wizytówce w confluence.
- Tworzymy bazę danych `` `pandopolis` `` wykonując zapytanie: `CREATE DATABASE `pandopolis`;`
- Ściągamy na swój komputer schemę bazy danych:
  `git clone git@dev.git.gd:cerebro/db_schema.git db_schema`
- Poprzez phpMyAdmina importujemy schemę z pliku `GAME_web.sql` do bazy `` `pandopolis` ``.
- Tworzymy bazę danych `` `pandopolis_servers` `` wykonując zapytanie: `CREATE DATABASE `pandopolis_servers`;`
- Edytujemy ściągnięty plik `GAME_servers.sql`. W ostatniej linijce zmieniamy 2 razy ciąg `"[game_db]"` na `"pandopolis"`.
- Importujemy schemę z ww. pliku do bazy `` `pandopolis_servers` ``.

4.2. Inicjalizacja wirtualki **`solid`**.

Czynności do wykonania na wirtualce `solid.pandopolis.gd`:

- Aktualizacja wirtualki. Poniższe kroki docelowo będą zbędne, gdy te zmiany zostaną zaszyte w samym obrazie. Możliwe, że były już na nim wykonane.
  - Login jako użytkownik `root` (hasło na stronie-wizytówce):
    1. `ln -s /usr/lib64/libboost_thread-mt.so /usr/lib64/libboost_thread.so`
    2. `yes | yum install telnet gdb git`
    3. `yum update`
  - Login jako użytkownik `servers` (hasło na stronie-wizytówce). Ściągnięcie kluczy umożliwiających dostęp gita do `dev.git.gd`.
    - `mkdir -p ~/.ssh`
    - `chmod 700 ~/.ssh`
    - `cd ~/.ssh`
    - `scp servers@solid.slots.gd:/home/servers/.ssh/config .`
      `scp servers@solid.slots.gd:/home/servers/.ssh/devgitgd_ci .`
      Hasło takie samo jak użytkownika `servers` na naszej instancji.
- Pobieramy najnowszy kod serwerów SGS i kompilujemy go.
  - Login jako użytkownik `servers`:
    - `cd`
    - `scp servers@solid.slots.gd:/home/servers/get_platform_src.sh .`
      Hasło takie samo jak użytkownika `servers` na naszej instancji.
    - `./get_platform_src.sh`
    - `cd /opt/servers/src ; ./Clean ; make`
      Cierpliwie czekamy aż serwery się skompilują, może to potrwać ~5-10 minut.

    Docelowo wszystkie powyższe kroki będą realizowane taskiem jenkinsowym o nazwie "Cerebro SGS Platform Release local".
    Będzie on parametryzowany nazwą brancha (default: `solid`) i adresem wirtualki.
- Konfigurujemy platformę SGS.
  - `cp /opt/servers/src/GlobalConf/server.template.cfg /opt/servers/bin/GlobalConf/server.cfg`
  - Edytujemy plik `/opt/servers/bin/GlobalConf/server.cfg`:
    1. wszędzie gdzie jest placeholder `PANDOPOLIS` wpisujemy prawidłową nazwę gry/ścieżkę
    2. na serwerze `solid.pandopolis.gd` upewniamy się, że w sekcji Misc mamy `Instance=2` (taki powinien być default).
    3. upewniamy się, że namiary na bazę danych podane w sekcji SQL są poprawne

    > Szczegółowy opis parametrów konfigurowalnych w pliku `server.cfg` znajduje się na stronie 1. Game

Config - opis zawartości pliku server.cfg.
Warto zwrócić uwagę zwłaszcza na wyjaśnienie czym są numery instancji platformy: Instancje serwerów.

W tym momencie mamy na wirtualce gotową platformę serwerową.

- Pobieramy przykładowy kod logiki gry. Będzie to bardzo prosta logika z kilkoma komendami, służąca jedynie do przetestowania działania serwera gry.
  - `cp -R /opt/servers/src/logic_example/* /opt/servers/src_logic`
  - `cd /opt/servers/src_logic ; make clean ; make`
- Edytujemy raz jeszcze plik `/opt/servers/bin/GlobalConf/server.cfg` ustawiając w nim tymczasowo ścieżkę do logiki 'example' w sekcji `[Cpp]`:
  - `LibPath=/opt/servers/bin_logic/logic_example/libLogicExample.so`
- Startujemy serwer gry:
  - `cd /opt/servers/bin`
  - `./nsc start all`

Opis struktury katalogów platformy SGS oraz krótki opis tego co jest we wzorcowym `Makefile` i dlaczego tak jest zrobione (struktura `bin_logic`, `build_logic`.) znajdą się na osobnej stronie, której jeszcze nie ma.
TODO - zrobić taką stronę.

W tym momencie mamy na wirtualce `solid` uruchomioną platformę SGS z numerem instancji równym 2.

Weryfikacja:

- Jako użytkownik servers wykonujemy:
  - `cd /opt/servers/bin ; ./nsc status all`

  Powinniśmy zobaczyć:
  ```
  Server ServerGame is running
  Server ServerBackend is running
  ```
  Jeśli któryś z serwerów nie jest w stanie `running`, to patrzymy do najnowszych logów w katalogu `/media/ephemeral0/logs` i szukamy komunikatów wskazujących na potencjalne problemy.
- Wchodzimy przez phpMyAdmina na `pma2.gd` do bazy `` `pandopolis_servers` ``.
- Sprawdzamy czy w tabeli `` `rtm_service_heartbeat` `` znajdują się wpisy z w miarę aktualną wartością timestampu w kolumnie `` `last_beat` ``.
- Sprawdzamy czy w tabeli `` `rtm_group_heartbeat` `` znajdują się wpisy z w miarę aktualną wartością timestampu w kolumnie `` `last_beat` `` oraz ze statusem `ready`.
  Jeśli brak jest grup w stanie ready, to patrzymy do najnowszych logów w katalogu `/media/ephemeral0/logic_logs` i szukamy komunikatów wskazujących na potencjalne problemy.
- Korzystając z testera logiki weryfikujemy poprawność działania logiki:
  - Wchodzimy na stronę `http://cerebro.gd/sgstester/`.
  - Wpisujemy namiar na nasz serwer: `solid.pandopolis.gd` (nie musimy podawać portu, default to `17000`), id użytkownika (jakaś wybrana mała liczba naturalna, np. 1) i dajemy "CONNECT!".
    Status połączenia powinien zmienić się na zielony napis "READY".
  - Wysyłamy do serwera kolejno komendy:
    - **`utils/load_config`** - w odpowiedzi powinniśmy otrzymać przykładowy global_config.
    - **`game_state/load`** - w odpowiedzi powinniśmy otrzymać przykładowy gamestate gracza
    - **`table/game_start`** - `game_state.table.game_in_progress` powinno zmienić się na 1
    - **`table/game_finish`** + w parametrach **`{"score": 5000}`** - w gamestate powinny zostać zaktualizowane m.in. wartości `games_completed`, `xp` i `high_score`.
  - Weryfikujemy poprawność działań na logice poprzez komendy konsoli:
    - Odpalamy jakiś klient telnet, np. putty. Łączymy się na host `solid.pandopolis.gd` i port `17100` (w przypadku puttyego ustawiamy Connection type na "Raw").
    - Wpisujemy poniższą komendę, gdzie X to id użytkownika, które podaliśmy przy podłączaniu:
      `logic message gamestate_id=X command=test/go params={} group=1`

      W odpowiedzi powinniśmy otrzymać:
      `Operation complete.`
      Natomiast w oknie testera logiki powinniśmy otrzymać komunikat `test/go` z parametrami `{ result: 11 }`.
  - Rozłączamy się z serwerem dając "DISCONNECT!".
    Status połączenia powinien zmienić się na czerwony napis "DISCONNECTED".

4.3. Inicjalizacja wirtualki **chaos**. Krok opcjonalny.

Cel:

- Posiadanie środowiska developerskiego, na którym developerzy mogą testować nowe featury, bez ingerencji w środowisko `solid`.
  Środowisko `solid` może być używane np. przez testerów, więc dobrze gdy jest tam zawsze jakaś działająca wersja gry, nie narażona na ciągłe restarty i bugi.
  Od tego może być właśnie `chaos`.

Czynności do wykonania na wirtualce `chaos.pandopolis.gd`:

- Czynności są dokładnie analogiczne jak na wirtualce `solid.pandopolis.gd` - są to bliźniacze serwery.
  Jedyną różnicą jest ustawienie podczas konfiguracji platformy SGS w pliku `server.cfg` wartości `Instance=3`.

4.4. Postawienie platformy SGS na komputerze lokalnym developera.

Cel:

- Developer logiki może kompilować, uruchamiać i debugować kod logiki na własnym komputerze.

Czynności do wykonania:

- Instrukcja stawiania platformy SGS na maszynie z Windows 7 opisana jest na stronie: Stawianie lokalnej instancji serwera pod Windows 7

Weryfikacja:

- Wchodzimy na stronę testera logiki `http://cerebro.gd/sgstester/`.
- Wpisujemy namiar na naszą instancję SGS: jako host nasze IP, reszta analogicznie jak przy weryfikacji wirtualki `solid`.
- Powinniśmy być w stanie wykonać te same czynności co na środowisku `solid`.

5. Przygotowanie środowiska umożliwiającego wystawianie klienta gry.

Cel:

- Zespół po utworzeniu nowej wersji klienta gry jest w stanie dostarczyć go w miejsce, z którego będzie mógł zostać osadzony w integracjach bazujących na środowisku lokalnym.

Czynności do wykonania:

- Login na wirtualkę `web.pandopolis.gd` jako użytkownik `root` (hasło na stronie-wizytówce):
  - Instalujemy narzędzia, których może nie być obrazie:
    - `yes | yum install mc`
  - `mkdir -p /opt/www/www-social/www`
  - Konfigurujemy apache tak, żeby adresy `web.pandopolis.gd` i `www.pandopolis.gd` wskazywały na ww. katalog:
    - Edytujemy plik `/usr/local/apache2/conf/extra/httpd-vhosts.conf` dopisując w nim na końcu:

```
<VirtualHost *:80>
        DocumentRoot /opt/www/www-social/www/
        ServerName web.pandopolis.gd
        ServerAlias www.web.pandopolis.gd
        ErrorLog
/usr/local/apache2/logs/www.web.pandopolis.gd-error-log
        CustomLog
/usr/local/apache2/logs/www.web.pandopolis.gd-access-log
combined
        <Directory /opt/www/www-pandopolis/www/>
                Options FollowSymLinks
                AllowOverride All
                Order allow,deny
                Allow from all
        </Directory>
</VirtualHost>
```

        - Weryfikujemy czy konfiguracja jest ok i restartujemy apache:
          `cd /etc/init.d && ./httpd configtest && ./httpd restart`
    - Tworzymy katalog, do którego będą wrzucane najnowsze wersje klienta gry i assety używane przez tego klienta:
        - `mkdir -p /opt/ganymede_upload/pandopolis/static/swf/`

      Sam podkatalog `swf` jest już opcjonalny, ale jeśli klient gry jest we flashu, to tak przyjęło się ten katalog nazywać.
    - Podmontowujemy ww. katalog, tak aby był dostępny przez Apacha:
        - `ln -s /opt/ganymede_upload/pandopolis/static/swf /opt/www/www-social/www/swf`

      Od tego momentu, to co wrzucimy do tego katalogu będzie dostępne pod adresem `web.pandopolis.gd/swf/`.
    - Potrzebujemy jeszcze na tej wirtualce umieścić plik `crossdomain.xml`, aby klient flash mógł się do niej odwoływać:
        - `vim /opt/www/www-social/www/crossdomain.xml`
        - Wrzucamy tam zawartość:

```
<?xml version="1.0"?>
<!DOCTYPE cross-domain-policy
        SYSTEM
"http://www.macromedia.com/xml/dtds/cross-domain-policy.dtd
">
<cross-domain-policy>
        <allow-access-from domain="*" to-ports="*"/>
</cross-domain-policy>
```

Weryfikacja:

- Login na wirtualkę `web.pandopolis.gd` jako użytkownik `root`:
    - `echo "Client placeholder." > /opt/ganymede_upload/pandopolis/static/swf/client.txt`
- Otwieramy w przeglądarce adres:
    - `http://web.pandopolis.gd/swf/client.txt`

  Powinniśmy zobaczyć wcześniej wpisaną treść pliku: "Client placeholder."
- Usuwamy zbędny plik:
    - `rm -f /opt/ganymede_upload/pandopolis/static/swf/client.txt`
- Otwieramy w przeglądarce adres:
    - `http://web.pandopolis.gd/crossdomain.xml`

  Powinniśmy zobaczyć treść pliku, którą wrzuciliśmy na serwer.

> Wskazówki nt. tego jak stworzyć przykładowego klienta gry, znajdują się na stronie: TODO(zrobić taką stronę):
>
> - Przykładowy prosty klient, albo flash, albo html5 (logic_tester).
> - Informacja o `client_template` dla flash. 1. Klient flash - moduł 'client_template'
> - Informacja o parametrach przekazywanych do klienta (`flashVars` w przypadku flasha). Flashvarsy

W tym momencie zespół może prowadzić development logiki i klienta gry.

Klienta developer może odpalać z własnego komputera, na którym prowadzony jest development, podając mu samemu odpowiednie parametry (namiary na serwer gry itp.).

6. Zainicjowanie repozytoriów z kodem.

Cel:

- Zespół prowadzi development kodu logi i klienta trzymając kod w repozytoriach dostępnych poprzez `dev.git.gd`.

Założenie:

- Mamy działające konto na firmowym gitlabie pod adresem `dev.git.gd`, na które jesteśmy w stanie się zalogować. Jesteśmy w stanie sklonować na swój komputer firmowe repozytoria git.
  Instrukcja zakładania konta znajduje się pod adresem: Internal git server

Czynności do wykonania:

- Utworzenie repozytorium kodu logiki:
  - Logujemy się na http://dev.git.gd.
  - Wchodzimy na adres http://dev.git.gd/groups/new i tworzymy grupę o nazwie takiej jak nazwa kodowa gry, czyli `pandopolis`.
  - Wchodzimy na adres http://dev.git.gd/groups/pandopolis/members i dodajemy wszystkich członków zespołu jako "Developer".
  - Klikamy "New project" i tworzymy w ww. grupie projekt o nazwie `logic`.
    Checkbox "Make project visible to everyone" zostawiamy zaznaczony.
  - Do zainicjowania repozytorium wygodnie jest użyć przykładowej minimalnej logiki, która testowana była na wirtualce `solid`.
    Dzięki temu będziemy mogli łatwiej przetestować w następnym punkcie poprawność działania procesu CI.
    W tym celu na swoim komputerze wykonujemy poniższe komendy. Przykładowe komendy są z linii poleceń Windows, dla linuxa będą analogicznie.
    - ```
      git clone -b solid git@dev.git.gd:cerebro/sgs.git sgs
      xcopy /S /Y sgs\logic_example logic_example\
      cd logic_example
      git init
      git add *
      git commit -m "Inicjalizacja repozytorium."
      git remote add origin git@dev.git.gd:pandopolis/logic.git
      git push -u origin master
      git checkout -b solid
      git push -u origin solid
      ```
    Po wykonaniu ww. komend będziemy mieli zainicjalizowane repozytorium z przykładowym kodem logiki.
    Ustawione będą branche `master` oraz `solid`. Ich użycie jest wskazane ze względu na stosowanie w projektach modelu developmentu opisanego na stronie Instancje serwerów.
    Przykładowy kod zawiera również pliki projektowe dla Visual C++, aby ułatwić start developmentu na maszynie Windows.
- Udostępnienie repozytorium członkom zespołu:
  - Standardowo dostęp do sklonowania repozytorium mają wszyscy użytkownicy. Aby móc do repozytorium pushować musimy dodać wybranych użytkowników do naszego projektu.
    W tym celu:
    - Wchodzimy na stronę `http://dev.git.gd/pandopolis/logic/team`.
    - Dajemy "New Project Member" i dodajemy członków zespołu jako "Developer", ew. jako "Master" jeśli chcemy, żeby sami również mogli zarządzać dostępem do repozytorium.

- Utworzenie repozytorium kodu klienta:
  - Tworzymy projekt analogicznie jw., tylko o nazwie `client`, zamiast `logic`.
  - Na ten moment nie mamy żadnego templejtu klienta inicjujemy więc puste repozytorium, zawierające przykładowy plik `client.bin`.
    - ```
      md client
      cd client
      git init
      echo "Client placeholder." > client.bin
      git add client.bin
      git commit -m "Inicjalizacja repozytorium."
      git remote add origin git@dev.git.gd:pandopolis/client.git
      git push -u origin master
      git checkout -b solid
      git push -u origin solid
      ```
  - Po wykonaniu ww. komend będziemy mieli zainicjalizowane puste repozytorium z przeznaczeniem na kod klienta gry.
- Udostępnienie repozytorium członkom zespołu:
  - Wchodzimy na stronę `http://dev.git.gd/pandopolis/client/team`.
  - Możemy albo dodać członków zespołu po kolei, albo skorzystać z opcji "Import members" i skopiować ich z projektu "`pandopolis / logic`".

Weryfikacja:

- Wchodzimy na stronę: `http://dev.git.gd/pandopolis/logic/tree/solid`.
  Powinniśmy zobaczyć listę plików przykładowej logiki.
- Wchodzimy na stronę: `http://dev.git.gd/pandopolis/client/tree/solid`.
  Powinniśmy zobaczyć plik `client.bin`, który umieściliśmy wcześniej w tym repozytorium.
- Prosimy kolegę/koleżankę z zespołu o sklonowanie repozytorium logiki na swój komputer i przetestowanie podstawowej funkcjonalności. Przykładowo:
  - Utworzenie nowego brancha na podstawie `solid`, wrzucenie na niego jakiegoś losowego pliku, push zmian na `dev.git.gid`, usunięcie brancha lokalnie i z `dev.git.gd`.

  Wszystkie te kroki powinny się zakończyć sukcesem.

git i GitLab (dev.git.gd) znacząco wpływają na przyjemność pracy z kodem, dlatego używanie tych narzędzi stało się w firmie de facto standardem.
GitLab posiada świetną funkcjonalność w postaci możliwość robienia merge requestów, co jest bardzo wygodne podczas robienia code review.
Umożliwia on wygodne przeglądanie diffa i umieszczanie komentarzy w dowolnym miejscu w kodzie.
Dlatego też polecanym modelem pracy jest robienie featury branchy -> implementacja zmian -> utworzenie merge requesta w GitLab -> code review (do skutku) -> akceptacja merge requesta + usunięcie brancha.
Jeśli jeszcze nie pracowałeś w takim modelu spróbuj, a zrobisz tym sobie dużą przysługę na przyszłość.

7. Ustawienie procesu continuous integration.

Cel:

- Kod logiki i klienta, który wrzucany jest na odpowiednie branche, jest automatycznie budowany i wystawiany:
  - Logika: branch `solid` wystawiany na `solid.pandopolis.gd`.
  - Klient: branch `solid` wrzucany do odpowiedniego katalogu na `web.pandopolis.gd`.

Do obsługi procesu continuous integration w firmie korzystamy z narzędzia Jenkins zainstalowanego pod adresem `jenkins.gd` .

Czynności do wykonania:

- Aby Jenkins mógł wgrywać pliki na wirtualki i wykonywać na nich operacje musimy najpierw dodać jego klucz do znanych kluczy SSH:
  - Logujemy się na wirtualkę `solid` na użytkownika `servers`.
  - `vim ~/.ssh/authorized_keys`
  - Wrzucamy do pliku treść:

```
ssh-rsa
AAAAB3NzaC1yc2EAAAABIwAAAQEA62meUmbTaDv21VMDhsHoddIwsJNS5tfp4ntu
7bOqjQdpksq2z2DzV76sWNUFgeYwPu57Wz+0QAMvhulhpBxfC9W7UOqq2PGx++Im
vb2VTkvimZbVh9IwCorwBzVO2tFm24ctCl/jh8dEZ5tgiO87ANRijnyzRnVigBAI
AzekbckzFMw1+sUCOOtAmu1V7tAp0gaFXB3oX4coU45S55kRcu84qoE8LBmtbW6m
EmCWdliKMhOsiZjNKUo5oBwK+1vtrl6rBJmJOaFBfeY7aXQNxcsPuMa8Q4CpXQJV
ylnNVypOwdAdzND91ZeEuhWg2622xx5xlhbbbYJKivRT44XIBQ==
jenkins@jenkins.gd
```

  - `chmod 600 ~/.ssh/authorized_keys`
  - Analogiczną operację wykonujemy na wirtualce `chaos`.
  - Analogiczną operację wykonujemy na wirtualce `web`, przy czym po zalogowaniu jako `root`, przechodzimy na użytkownika `www`: `su - www`.
- Jeśli jeszcze nie mamy konta na `jenkins.gd`, to je zakładamy pod adresem http://jenkins.gd/signup.
- Logujemy się swoim kontem na `jenkins.gd`.
- Tworzymy nową zakładkę, gdzie będą umieszczone taski dot. naszego projektu:
  - Wchodzimy na stronę http://jenkins.gd/newView.
  - Jako nazwę projektu wpisujemy jego pełną nazwę, nie musi być kodowa, np. `Pandopolis`.
  - Wybieramy "List View".
- Praktyka wykazała, że do wystawiania logiki wygodnie jest mieć dwa taski: jeden do wystawiania zmian tylko w global configu, a drugi do wystawiania całego kodu logiki.
  W ten sposób wystawienie zmian tylko w global configu jest dużo szybsze.
  Poniższy opis bazuje na istniejących taskach dot. LastTemple. W przypadku gdyby tych tasków już nie było, możemy skorzystać z innego projektu - wszędzie taski powinny być analogiczne.
- Zaczynamy od taska pierwszego:
  - Przechodzimy na zakładkę Pandopolis i tworzymy nowe zadanie: http://jenkins.gd/view/Pandopolis/newJob.
  - Jako nazwę podajemy: "Pandopolis Logic Reload on solid.pandopolis.gd - branch solid".
  - Wybieramy opcję "Kopiuj z istniejącego Job" i jako źródło podajemy task "LastTemple LogicReload on solid.lasttemple.gd - branch solid".
  - Wszędzie gdzie jest wpisany LastTemple zmieniamy nazwę/ścieżki na te wskazujące na `pandopolis`. Będą to pola:
    - Opis
    - Source Code Management / Git / Repository URL

        - Build / Execute Shell / Command (są dwie komendy)
    - Dodatkowo jeśli nasz plik z global configiem ma inną nazwę niż `global_config.json` to podajemy ją w polu: Source Code Management / Git / Branches to build / Zaawansowane / Included Regions.
- Tworzymy task aktualizujący cały kod logiki:
    - Przechodzimy na zakładkę Pandopolis i tworzymy nowe zadanie: http://jenkins.gd/view/Pandopolis/newJob.
    - Jako nazwę podajemy: "Pandopolis Logic Release to solid.pandopolis.gd - branch solid".
    - Wybieramy opcję "Kopiuj z istniejącego Job" i jako źródło podajemy task "LastTemple Logic Release to solid.lasttemple.gd - branch solid".
    - Wszędzie gdzie jest wpisany LastTemple zmieniamy nazwę/ścieżki na te wskazujące na `pandopolis`. Będą to pola:
        - Opis
        - Source Code Management / Git / Repository URL
        - Build / Execute Shell / Command
        - Post-build Actions / Build other projects - ustawiamy wcześniej utworzony task "Pandopolis Logic Reload on solid.pandopolis.gd - branch solid".
    - Dodatkowo jeśli nasz plik z global configiem ma inną nazwę niż `global_config.json` to podajemy ją w polu: Source Code Management / Git / Branches to build / Zaawansowane / Excluded Regions.
- Tworzymy task wystawiający klienta na `web.pandopolis.gd`:
    - Przechodzimy na zakładkę Pandopolis i tworzymy nowe zadanie: http://jenkins.gd/view/Pandopolis/newJob.
    - Jako nazwę podajemy: "Pandopolis Client Release to web.pandopolis.gd - branch solid".
    - Wybieramy opcję "Build a free-style software project".
    - Wpisujemy:
        - Opis: `"Buduje klienta flash z brancha solid z dev.git.gd:/pandopolis/client i wystawia lokalnie na web.pandopolis.gd."`
        - Source Code Management / Git / Repository URL: `git@dev.git.gd:pandopolis/client.git`
        - Source Code Management / Git / Branches to build / Branch specifier: `solid`
        - Build Triggers: `Poll SCM`, Schedule: `* * * * *`.
        - Build: Tutaj wpisujemy cokolwiek pozwoli nam zbudować naszego klienta z kodu wystawionego z gita. Przykładowo dla projektów flash sugerowaną metodą jest wpisanie wszystkich kroków do skryptu ant. Dajemy wówczas "Add build step" / "Invoke Ant" i ustawiamy "Ant version": `ant` i "Targets": `"NAZWA TARGETU"` (taki jaki mamy wpisany w skrypcie ant, który zbuduje klienta).
        - Build: "Add build step" / "Execute shell" i wpisujemy komendę:
          ```
          rsync -c -v -r -z --exclude '.git' --rsh='ssh -o StrictHostKeyChecking=no' "${WORKSPACE}/bin-release/" "www@web.pandopolis.gd:/opt/ganymede_upload/pandopolis/static/swf"
          ```
          Zamiast `"${WORKSPACE}/bin-release/"` możemy podać inną ścieżkę - taką, do której zostanie wrzucony zbudowany klient.
          Również wrzucanie do podkatalogu `swf` jest opcjonalne - ale dla klientów flash taka przyjęła się u nas konwencja.

> Wersjonowanie klienta gry.
>
> Powyższy opis nie wspomina o wersjonowaniu klienta gry. Ten temat jet poruszony w innym dokumencie. (TO DO - dać tu link jeśli strona z taką informacją powstanie - wskazówki dot. tworzenia klienta)
> W momencie gdy zaczniemy klienta wersjonować w tym tasku pojawią się dodatkowe kroki.

Weryfikacja:

- Aktualizujemy kod logiki gry i pushujemy zmianę na branch `solid`:
    - Jeśli mamy do repozytorium wgrany kod logiki `logic_example` to możemy zmienić przykładowo wartości wyświetlane w funkcji `GLogicEngineExample::handle_system_message()` podczas obsługi komunikatu z konsoli `test/go`.
      Zamiast `"output = "Operation complete.";"` dajemy np. `"output = "Operation done!";"`
    - Pushujemy zmianę do gita i czekamy ok. 1 minutę.
    - Wchodzimy na `jenkins.gd` na projekt "Pandopolis Logic Release to solid.pandomania.gd - branch solid" i patrzymy czy pod "Build History" pojawił nam się nowy build.
    - Odpalamy jakiś klient telnet, np. putty. Łączymy się na host `solid.pandopolis.gd` i port `17100` (w przypadku puttyego ustawiamy Connection type na "Raw").
    - Wpisujemy poniższą komendę:
      ```
      logic message gamestate_id=1 command=test/go params={} group=1
      ```
      W odpowiedzi powinniśmy otrzymać nowy komunikat:
      ```
      Operation done!
      ```
- Aktualizujemy kod klienta i pushujemy zmianę na branch `solid`:
    - Dodajemy do repozytorium plik `jenkins_test.txt` z dowolną zawartością, np. `"Jenkins is working!"`.
    - Pushujemy zmianę do gita i czekamy ok. 1 minutę.
    - Wchodzimy na `jenkins.gd` na projekt "Pandopolis Client Release to web.pandopolis.gd - branch solid" i patrzymy czy pod "Build History" pojawił nam się nowy build.

- Wchodzimy przeglądarką pod adres: `http://www.pandopolis.gd/swf/jenkins_test.txt`
  Powinniśmy zobaczyć zawartość naszego wcześniej utworzonego pliku.
- Usuwamy plik `jenkins_test.txt` z repozytorium.

8. Kroki opcjonalne.

Cel:

- Ułatwienie dalszej pracy członkom zespołu nad developmentem gry.

8.1. Identyfikacja wirtualek.

Cel:

- Natychmiastowe orientowanie się, na który serwer jesteśmy zalogowani w danym okienku putty.

Czynności do wykonania:

- Po kolei na writualkach `chaos`, `solid`, `web`:
  - Logujemy się jako `root`.
  - Wchodzimy na stronę http://www.network-science.de/ascii/ i generujemy napis odpowiedni do bieżącej wirtualki: "`chaos.pandopolis`", "`solid.pandopolis`", "`web.pandopolis`".
    Sugerowane jest użycie fonta o nazwie "big". Jeśli napis nie mieści się w 80 kolumnach, to go skracamy albo rozbijamy na 2 linijki.
  - `vim /etc/motd` i wklejamy tam wygenerowany napis.

Weryfikacja:

- Prosimy kolegę/koleżankę z zespołu o zalogowanie się na dowolnie wybraną wirtualkę.
- Podchodzimy i sprawdzamy czy od razu widać, która wirtualka została wybrana.

8.2. Dostęp do wirtualek poprzez Sambę.

Cel:

- Przyspieszenie procesu wrzucania i edytowania plików na wirtualkach.
  Dzięki bezpośredniemu dostępowi do wirtualki z poziomu maszyny Windows możemy m.in. operować wygodnie na repozytorium git fizycznie znajdującym się na wirtualce, tak jakby było u nas na komputerze.

Czynności do wykonania:

- Prosimy dział OPS o wpięcie wirtualek do domeny `GANYMEDE`.
- Na wirtualkę `solid` logujemy się jako `root`.
  - `yes | yum install samba`
  - `vim /etc/samba/smb/conf`
    Podmieniamy całkowicie zawartość pliku na poniższą:

```
[global]
workgroup = GANYMEDE
server string = solid.pandopolis.gd
hosts allow = 192.168. 127.

interfaces = 192.168.0.134

log file = /var/log/samba/log.%m
max log size = 500
log level = 0
security = domain
passdb backend = tdbsam
```

```
password server = *
idmap uid = 10000-20000
idmap gid = 10000-20000
winbind separator = +
winbind use default domain = yes
winbind enum users = yes
winbind enum groups = yes
auth methods = winbind

domain master = no
domain logons = no
local master = no
preferred master = no

wins support = no
wins server = 192.168.0.10
load printers = no

#cups options = raw
load printers = no
printing = bsd
printcap name = /dev/null
show add printer wizard = no
disable spoolss = yes

create mask = 0664
directory mask = 0775

template shell = /bin/bash

unix extensions = no
notify:inotify = false
deadtime = 15
socket options = TCP_NODELAY IPTOS_LOWDELAY SO_RCVBUF=65536
SO_SNDBUF=65536
strict sync = no
strict locking = no
sync always = no
getwd cache = yes
max open files = 100000

[servers]
    path = /opt/servers/
    public = no
    writable = yes
    printable = no
    delete readonly = yes
    create mask = 0664
    directory mask = 0775
```

```
        force user = servers
        force group = servers
        valid users = @GANYMEDE+social_dev
```

    - Edytujemy linijki:
      `server string = solid.pandopolis.gd`
      `interfaces = 192.168.0.134`
      W pierwszej ustawiamy odpowiednią nazwę projektu. W drugiej podajemy IP wirtualki z sieci 192.168.0.*. Możemy go sprawdzić komendą `/sbin/ifconfig` - prawdopodbnie będzie to interfejs `eth1`.
    - `cd /etc/init.d`
    - `./smb restart && ./nmb restart && ./winbind restart`
    - W tym momencie powinniśmy mieć udostępniony przez Sambę katalog `/opt/servers/`.

- Na wirtualkę `web` logujemy się jako `root`.
    - `yes | yum install samba`
    - Dokonujemy czynności analogicznych jak dla wirtualki `solid`, przy czym zamiast sekcji `[servers]` na końcu wstawiamy sekcję:

```
[static]
    path = /opt/ganymede_upload/pandopolis/static/
    public = no
    writable = yes
    printable = no
    delete readonly = yes
    create mask = 0664
    directory mask = 0775
    force user = www
    force group = www
    valid users = @GANYMEDE+social_dev
```

    - Po zrestartowaniu Samby, powinniśmy mieć udostępniony z wirtualki katalog `/opt/ganymede_upload/pandopolis/static/`.
- Na wirtualce `chaos`, w razie potrzeby, powtarzamy kroki z wirtualki `solid`.

Weryfikacja:

- Na maszynie Windows w oknie "Mój Komputer" próbujemy wejść pod adres `\\solid.pandopolis.gd\`. Wchodzimy do udziału `servers`.
  Powinna nam się ukazać zawartość katalogu `/opt/servers/`.
- Na maszynie Windows w oknie "Mój Komputer" próbujemy wejść pod adres `\\web.pandopolis.gd\`. Wchodzimy do udziału `static`.
  Powinna nam się ukazać zawartość katalogu `/opt/ganymede_upload/pandopolis/static/` (m.in. podkatalog `swf`, jeśli go tam utworzyliśmy).

8.3. Panel do generowania global configu.

Cel:

- Ułatwienie i przyspieszenie procesu dokonywania zmian w global configu gry. Uniknięcie pomyłek przy ręcznej jego edycji.
  Kontrola, aby najnowsza wersja global configu była zawsze w kontroli wersji.

Czynności do wykonania:

- Dokument ten nie zawiera instrukcji tworzenia nowego panelu. Warto przed jego utworzeniem spojrzeć na istniejące panele do gier:
  `panel.slots.gd`, `panel.lasttemple.gd` itd.

- Wytyczne jakich ten panel powinien się trzymać to:
  - Trzymanie swoich danych w bazie o nazwie `pandopolis_config_panel`.
  - Kod panelu trzymany w repozytorium pod adresem `http://dev.git.gd/pandopolis/global_config_panel`.
  - Task w jenkinsie o nazwie "Pandopolis GlobalConfig Panel Release (branch master)" automatycznie wystawiający kod panelu pod adres `http://panel.pandopolis.gd`.
  - Logowanie do panelu powinno odbywać się kontem JIRA.
  - Powinien być możliwy z panelu eksport zmian do pliku global_config w repozytorium gry, zarówno na branch `solid` jak i `master` (do wyboru przez użytkownika).

Weryfikacja:

- Wchodzimy na adres `panel.pandopolis.gd`.
- Logujemy się swoim kontem JIRA.
- Dokonujemy testowej zmiany konfiguracji.
- Eksportujemy global config na branch `solid`.
  W tym momencie powinien uruchomić się m.in. task jenkinsa wrzucający nowy global config na wirtualkę `solid` i przeładowujący logikę.
  Na instancji `solid` powinna być dostępna nowa logika.
- Eksportujemy global config na branch `master`.
  W tym momencie powinna zostać utworzona nowa paczka z logiką, która wkrótce powinna być dostępna do wystawienia z poziomu panelu core na AWS.

8.4. Panel debugowy pod grą.

Cel:

- Możliwość wysyłania do logiki gry, w środowisku lokalnym, dodatkowych komunikatów z poziomu strony www, na której osadzona jest gra.
  Mogą to być komunikaty typu "włącz nieśmiertelność", które są znaczną pomocą podczas testowania gry.
- Możliwość wygodnego podłączania się do serwera gry uruchomionego na komputerze developera.

Czynności do wykonania:

- Dokument ten nie zawiera instrukcji tworzenia nowego panelu. Warto przed jego utworzeniem spojrzeć na istniejące panele debugowe do gier.
  W tym celu można się zgłosić do kolegów/koleżanek z innych zespołów.
- Wytyczne jakich ten panel powinien się trzymać:
  - Panel debugowy osadzony jest w ramach konkretnej integracji z wybranym sajtem. Dlatego też jego kod powinien być trzymany w tym samym repozytorium co kod integracji.
  - Panel powinien umożliwiać wybór serwera z jakim chcemy się połączyć (`solid`, `chaos`, instancja na komputerze developera).
  - Panel powinien umożliwiać wysyłanie w wygodny sposób (jakieś przejrzyste UI) wybranych komend do logiki gry, do której jesteśmy zalogowani.
    Mechanizm ten opiera się na wysyłaniu komend do serwera gry poprzez konsolę.

Weryfikacja:

- Wchodzimy na grę, na integrację, na której osadzony został panel debugowy.
- Możemy swobodnie przełączać się między instancjami `solid` i `chaos`, ew. inną instancją gry uruchomioną na komputerze developera z listy Instancje serwerów.
- Możemy wykonywać na logice operacje, standardowo niedostępne graczom.
  W szczególności panel powinien udostępniać operację typu "clear gamestate".

8.5. Panel do translacji.

Cel:

- Umożliwienie wygodnego dodawania nowych wersji językowych gry - w sposób już znany i przetestowany.
  Możliwość automatycznego buildu klienta gry z użyciem tekstów wprowadzonych w tym panelu.

Czynności do wykonania:

- Posiadamy u nas wewnętrzny panel do tłumaczenia tekstów, który jest rozwijany aktualnie przez zespół Cerebro.
  Z punktu widzenia product ownera, wygodne może być, żeby do obsługi tłumaczeń użyć tego właśnie panelu i flow jaki

on oferuje.
Tak, aby nie tworzyć za każdym razem nowego rozwiązania.
Przykładowo, panel ten zainstalowany jest pod adresem: `translation.slots.gd` (logowanie kontem JIRA).
- Instrukcja instalacji panelu tłumaczeń, jak każdego produktu Cerebro, znajduje się w confluence w sekcji Panel tłumaczeń.

Weryfikacja:

- Wchodzimy pod adres `translation.pandopolis.gd`.
- Logujemy się swoim kontem JIRA.
  Powinniśmy w tym momencie móc dodawać i edytować teksty z gry.