Victoria University of Wellington
School of Engineering and Computer Science

SWEN438: DevOps

Assignment 3 — Batch Size

Due: 5th September @ 23:59

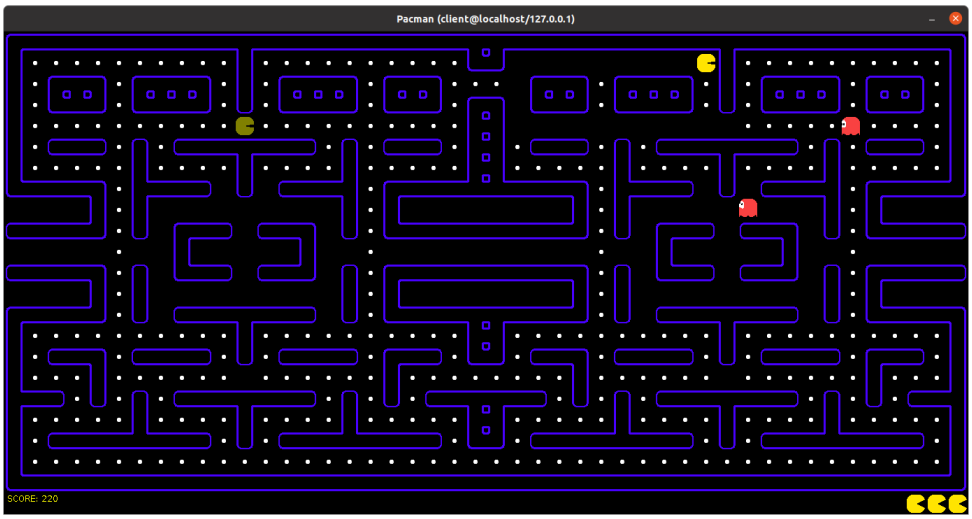
Overview

This assignment is focused on the *process* of developing software using *small batches*. In the assignment, you will be tasked with adding several features to an existing program (*multiplayer pac-man*). Unlike many assignments you may have done before, this assignment is more concerned with *how* you go about developing these features and *less* concerned with the final outcome. You will be using gitlab extensively during this assignment, with marks being awarded (or deducted) for each commit. For example, you will be marked down for *breaking the build* on the `main` branch. **Therefore, you will need to think carefully before pushing a commit to the `main` branch**.

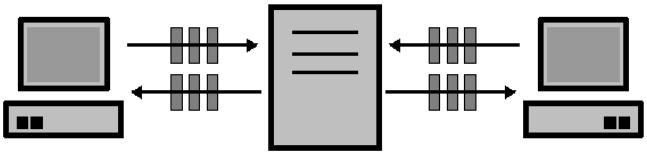
HINT: We suggest you familiarise yourself with GitMark on a temporary repository before beginning the project itself. See the Appendix for instructions.

Multiplayer Pac-Man

The original *Pac-Man* was developed in 1980 by Namco and released as an arcade machine (see the Wikipedia Page for more). In this assignment, you will work with a variation extended with multi-player support.



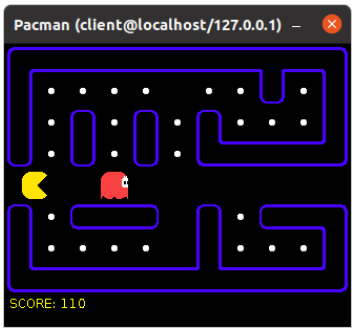
Here, we see a multiplayer game with two players (where the dimmed Pac-Man represents *another* player). To implement its multiplayer functionality, the program adopts a *client / server* architecture:



Here, we see two clients connected to a single *Pac-Man server*. Each client corresponds to one of the players in the game, and controls that player by sending commands to the server and receiving updates in return. To begin a new game, the server starts with a given *game board* and waits for the right number of clients to connect. Once enough players have connected, the game begins. If a player disconnects for some reason, they are removed from the board and cannot return to the game.

Game boards are given in a textual notation which is relatively easy to understand. The following illustrates a game board:

```
WWWWWWWWWWW
W....B..W.W
W.W.W.W...W
W.W.W.WWWWW
.....  .....
W.WWW.W.WWW
W....XW...W
WWWWWWWWWWW
```



Here, "`W`" indicates a *Wall* and "`.`" indicates a *Pill*, whilst "`X`" marks a *player portal* and "`B`" marks a portal for *Blinky*.

Small Batch Development

Reducing batch size is a key aspect of the DevOps approach to software development (e.g. as advocated in the *DevOps Handbook*). But, this raises an important question: *what does batch size mean in the context of software development?*. In this assignment, you will explore one (somewhat extreme) perspective on this question.

The multi-player Pac-Man program you have been given is missing several key features, including support for *power-up pills*, *different ghost strategies* and more. **There are at least five distinct features in total and you are tasked with implementing them all**. However, since this assignment is about process more than anything, some requirements are placed on you:

- **Commit Size**. Marks will be awarded for *small commits* and deducted for *large commits*. Thus, you must carefully plan your development cycles to ensure incremental progress towards each goal.
- **Builds.** Marks will be deducted for commits which *break the build*. That is, for commits which do not compile using `javac`. Initially, the code given to you compiles without problem.
- **Tests.** Marks will be deducted for commits which have tests that are reported as failing by GitMark. Initially, all tests are passing. However, a number are *disabled* using the JUnit 5 `@Disabled` attribute.[1] Therefore, to begin work on a feature you will uncomment one or more disabled tests. When the feature is complete you will be able to make a commit where that test is no longer disabled. However, in between, you may need to comment it out!

[1] Whilst GitMark considers these as passing tests, they represent features that are not yet working. *In the final grading for the assignment, only passing tests that are not disabled will be counted.*

In addition, marks will be awarded for correctly implementing the various features via *acceptance tests*. Marks are also available for the quality of *commit messages* and for using *issues* appropriately. Full details of the marking algorithm for commits are given below.

To help you with this process, a tool is provided (*GitMark*) which can mark a commit for you. This means that, before pushing a commit to the final repository, you can check whether it meets the above requirements.

NOTE: All commits reachable from the `HEAD` of `main` at the point of submission be marked according to the above requirements. Therefore, you may work on feature branches which are then tidied up (e.g. *rebased*) before being merged into `main`.

Submission

For this assignment, you must create a new repository entitled `Assignment_3` in your designated SWEN438 project area:

`https://gitlab.ecs.vuw.ac.nz/course-work/swen438/username/`

To make a submission for this assignment, you must submit a text file entitled "`submission.txt`" to the submission system. This file should contain a link to your gitlab repository. The submission system will check your repository is in the correct location and will automatically mark your assignment. **You must ensure your submission is accepted by the submission system without error**. Note, you are permitted to make as many submissions prior to the deadline as you wish.

Marking Criteria

The assignment will be assessed using both quantative and qualitative marking. The former will be marked using an automated marking system (GitMark), whilst the latter will be marked by hand. The breakdown of marks is as follows:

- **Correctness (40 marks)**. Overall correctness of implemented features will be judged according to a set of *acceptance tests*. These will be similar (though not necessarily identical) to those disabled tests provided with the Pac-Man implementation.
- **Commits (40 marks)**. Every commit will be worth at most **one mark**. In order to gain this mark, the commit must meet the following criteria:
  - **Size**. The commit must be less than 500 bytes in size, as judged by `GitMark`.
  - **Build & Test**. The commit must compile and pass all available tests.

  Observe that this means we are expecting around 40 commits! Furthermore, to eliminate "gaming" of the system, $\lfloor \frac{s}{1000} \rfloor$ marks will be deducted for commits ≥ 1000 bytes in size (where s is the commit size).
- **Commit Messages (10 marks)**. The manner in which you have used version control will also be assessed. Specifically, it is expected that all commits have sensible and coherent commit messages. As such, you are recommended to use a sensible style for your commit messages.[2]
- **Issue Tracking (10 marks)**. You are also expected to use the *GitLab issue tracker* in an appropriate fashion. In particular, it expected that every *feature* added to the program has *at least one issue* raised against it (and potentially many more). The issue description is expected to clearly and coherently described the issue (or feature) being addressed.

[2] See "How to Write a Git Commit Message", https://chris.beams.io/posts/git-commit/.

Appendix: Using GitMark

GitMark is a standalone tool for marking a commit history. For a given repository, it tracks back from the `HEAD` of the default branch (i.e. `main`) marking each commit in turn. Here is a sample output from the tool:

```
d106c757cbb58274d325ebb959db21c40a3c6374 ............................ [0 marks]
3a2c8b7d058fdb943540540e77fcad5d3acf3c72 ............................ [1 marks]
567b2ec8b850fe94e34fecc98c163f1927f3c7a9 ............................ [1 marks]
b79a699e91d14146905dd1689cea9f32e99ab1b8 ........................... [-3 marks]
d40324296e961b99337d2c5868e0caa124051eae ............................ [1 marks]
e6adf8a97359b9ed463e21f6b6ca91d6a769ffa3 ............................ [0 marks]
8975ef90f5e5a05d04363efcc42ec3fab73a158e ............................ [1 marks]
399eab6b419cd4c98c9d7a481fbc3a8a999154dc ............................ [1 marks]
90f46471ec9507fbcb826517f8ac6126c642e1d4 ............................ [1 marks]

================================================================================
COMMIT#0: d106c757cbb58274d325ebb959db21c40a3c6374                    0 marks
================================================================================
"Initial commit"

Java Build && Java Test? Yes.
First commit? Yes.
= 0 mark(s).

--------------------------------------------------------------------------------
Java Build:

--------------------------------------------------------------------------------
Java Test:

50 / 50 tests passed.

================================================================================
COMMIT#1: 3a2c8b7d058fdb943540540e77fcad5d3acf3c72                    1 marks
================================================================================
"Fix for Blinky targetting system"

Java Build && Java Test? Yes.
First commit? No.
Commit Size = 98 in
Commit Size < 500? Yes.
= 1 mark(s).

--------------------------------------------------------------------------------
Java Build:

--------------------------------------------------------------------------------
Java Test:

51 / 51 tests passed.
--------------------------------------------------------------------------------
Commit size:

src/pacman/server/characters/Blinky.java                             109 bytes
src/pacman/server/testing/SinglePlayerTests.java                     -11 bytes
                                                                   = 98 bytes
```

At the top, the tool illustrates the marks given for each commit. *Observe that marks are not awarded for the first commit*. After this we have a detailed output telling us what happened for each commit (e.g. whether the build failed, testing failed, and the size of the commit). For commit `#1` we can see that it passed the build and test stages, and was identified as a small commit (i.e. < 500 bytes); hence, a mark was awarded for the commit.

GitLab CI/CD. The easiest way to get started with GitMark is through the gitlab CI/CD pipeline. For example, you can use the following `.gitlab-ci.yml` file to get started:

```
image: whileydave/gitmark:latest

default:
  tags:
   - docker

build:
  stage: build
  script:
    - gitmark .
```

This uses the `gitmark` Docker image, and invokes `gitmark` on the current working directory (i.e. where your repository is checked out inside the GitLab runner). If you push this to your gitlab repository you should be able to see that it has succeeded, and what the output was. *At the current time, gitmark never causes the build to fail.*

NOTE: GitMark is quite particular about the location of the Java source files. They must be located in the `src/` directory within your repository (i.e. *not* in the Maven layout). You can also pass the command-line option `-last` to tell GitMark to consider only the last commit. This can speed up build times.

Command-Line Tool. There is also a command-line tool which you can download from the SWEN438 lecture schedule. This allows you to run GitMark on your local machine (which can be faster).

Appendix: Running Multiplayer PacMan

The multiplayer PacMan game provided can be run in either *single player* or *multiplayer* mode.

Single-Player Mode To run in single player mode, the following command suffices:

```
java pacman.server.Main boards/classic.txt
```

Here, the *classic* PacMan board is used, but you can use others. You may also prefer to do this from Eclipse (or IntelliJ) by setting up an appropriate *run configuration*.

Multi-Player Mode To run PacMan in multiplayer mode, the first thing is to start a server somewhere. The following command illustrates:

```
java pacman.server.Main -server 2 boards/classic.txt
```

This starts a server which is expecting *two* client connections. Therefore, to start playing we need to create two clients which connect to this server. We can create a client using the provided client jar as follows:

```
java -jar pacman-client-v1.0.jar servername
```

This creates a client which attempts to connect to `servername` (e.g. `barretts.ecs.vuw.ac.nz`). To connect to a server running on the same machine, use `localhost`. When both clients are connected to the server, the game will begin and will continue until all clients have disconnected.

Appendix: Ghost Targeting

Understanding the mechanics of ghost targeting is important for completing this assignment. You can find more information here:

- https://gameinternals.com/understanding-pac-man-ghost-behavior