Laboratory #10: Setup your team and project

Game Programming 1 (420-141-VA) - Fall 2020
Teacher: Tassia Camoes Araujo

Goals:
1. Confirm your team composition
2. Setup the project and practice git
3. Launch your Game Design Document

Instructions

Part I: Team composition and name

Check you team in the table below and talk to your team mates. If your name is not in this table, talk to me as soon as possible. Considering that everyone is happy with that team, discuss and decide the following:

1. Game name
2. Where the game will be hosted (under which service and username?)
3. Main communication platform for the team
   - Synchronous: chat or audio/video call, choose your platform
   - Asynchronous: zulip, MIO, email, choose and stick to it

Pitch id #	Teams	Game pitch
1	Vlad Cristian Dumitrescu Tivig Nicholas Ciobanu Alvin Alagos Eli YiuKai Wong	https://github.com/rol3293/final-project-idea
4	Alex Nguyen Kevin Judal	https://github.com/AliAlnajja/gameprojectidea
5	Christopher Andrada Ali Alnajjar Igor Raigorodskyi Carla Oliverio	https://github.com/ISellMangos/CrossingTheStreet
6	David Ano-Trudeau Theodore Tsimiklis George Athanasatos Anthony Nadeau	https://github.com/DavidAnoTrudeau/Game-Proposal
7	Elohe Darel Argot Kofi Osel	https://github.com/DarelA-Program/Game-Pitch
10	Chilka Castro José Samayoa	https://github.com/chilkacastro/GameProgLab8

	Nicolas Nguyen Madalina Turcan	
10	Shawn Gregory Hong Hien Pham Krikor Astour	https://github.com/chilkacastro/GameProgLab8
11	Curtis Chioda Douyon Sebastiampillai Dé Andre Powell McKain	https://github.com/CurtC187/Lab-8-Parts-2-and-3
13	Giuliana Bouzon Chelsea Nadarajah-Chinnia Andrei Marinescu	https://github.com/gbouzon/game-programming
16	Sarah Primavera Hawad Ahmad	https://github.com/chels1023/gameproglab8
18	Marc Neefa Abisan Poothapillai Gevorg Markarov	https://github.com/MarcNefaa/GamePitch
23	Ibrahim Awad Rana-Partap Kehal Muhammad Huzaifa Alam Khan	https://github.com/Abisan-AP/MyGame

Part II: Project setup

1. Once you've defined where the project is going to be hosted, one developer should create the repository (please, use the name of the game!) and add the other developers of the team as contributors with permission to write to the repository.
2. All team members should clone the repository and do a **first trial commit** (send any file, or edit a file already there. Make sure you use informative log messages in your commits.
3. One team member should create a Greenfoot project locally, and move the whole project folder to the directory containing the local git repository. Then use the commands **git add**, **git commit** and **git push** to send the project to the remote repository.
4. All the other team members should now update their local version of the project by entering the local repository and running the **git pull** command.
5. Now, one developer at a time:
   a) make a small change to the project, make sure it compiles and works, and again, **git add**, **git commit** and **git push**;
   b) check online that your changes were uploaded successfully;
   c) tell the rest of the team that your changes are up for them to retrieve (**git pull**) and test your code in their machine;
   d) Call the next one in the team to do the same.

You should continue this practice until all members of the team are comfortable with collaborating using git. Even if you are totally comfortable with it, having one team member that struggles with git might impact the whole team's work. Invest some time now to play with it, mess up, recreate the project, recreate the repository if needed. Better to make errors and learn from your errors at this point than later on in the project development.

Once the whole team has gone through the step 5 above, and everyone is comfortable with the process, try to create a conflict in code, and solve it. A conflict will occur if you try to push to the remote repository, but your changes are based in an older version than what is currently online. For instance, suppose the code online is on version 1.0. Everyone in the team has the same version in their local repository. Now two members of the team start making local changes at the same time, without communicating with the other. When the first one pushes to the repository, it will be fine, since the next version, 1.1, was created based on the 1.0 which is online. Now, when the second person tries to push, also a 1.1 version, based on the old 1.0, there will be a conflict with the current 1.1 already online.

As a rule of thumb, git can only push on top of a version that is on the history of versions of the code. The code that your team mate sent is totally foreign to your own local repository, it does not recognizes it as part of its history, that is the source of conflict.

Some conflicts are hard to solve, and it sometimes you'll need to manually fix things, or clean up files, to make it work. Git will be able to merge changes if they occur in different files, or even completely different portions of the same file. But if you and a team mate change the same java method, git cannot know which lines to keep, and which lines to discard. This is when the conflict needs to be solved manually.

Read the messages git output with attention, and it will tell you at which file to look for conflicts. You'll need to remove all lines inserted by git, until it is a clean java file.

So either you coordinate your changes very well, and never have a conflict, or you learn how to solve conflicts in git. If you divide your tasks well, and communicate with others about what you are doing, conflicts should not happen often.

Part III: Game Design Document (GDD)

The Game Design Document (GDD) is the team's main working document. It describes the team's vision for the game and should guide the development process. To make sure all developers share the same vision, please invest time in documenting decisions as the plan evolve, and avoid frustrations along the way.

The GDD is a "live" document that is frequently updated as the development progresses. Writing starts at the pre-production phase, but finishes only when the Golden Master is released.

1. Create the file to host your GDD in a public space.
    - ✔ I strongly suggest you use either Office 365 or Google Docs for this task, so that everyone can edit the file collaboratively. Avoid having different versions of the file floating around.

An industry-standard GDD is very technically detailed, and also take commercial issues into account such as budget. For this project, a simplified version is expected.

The following sections should be present:

- Executive summary: an overview of what the game is about, related games, the space in which it takes place (the World!), the main characters, enemies and allies, main game mechanics, characters goals, and game win/lose conditions. Provide images for each visual element, even if they are just placeholder images at this point.
- User Interface Mock-up(s): provide a visual representation of the game with progression of levels, user interface and flow between screens. It can be hand-drawn, drawn using a diagram software, or a collage of images.
- Controls: Describe how the game mechanics map to controller inputs.
- Developer roles: think of the different tasks and roles involved in the game development pipeline, and define the composition of your team based on those. Roles can be rotated, so that everyone will have a taste of each task, still respecting peoples profiles and skills. Assign team members' responsibilities for the prototype of the game (the first version to be delivered). Each week, responsibilities can be re-assigned.
- Schedule of releases: in each version of the game, describe the stage of development that you expect to achieve. Try to list all the tasks involved in the development, each features you want delivered, and distribute those among the versions:
  - 11/15: Prototype
  - 11/22: First playable
  - 11/29: Pre-Alpha
  - 12/06: Alpha
  - 12/10: Gold Master

Your game GDD will be evaluated on a weekly basis, along with the evolution of the game in the public repository. Do not forget to budget the time for documentation (GDD, README, LICENSE, and code comments), it should take some time on a regular basis.

Part IV: Deliverables

1. Only one lab report is needed per team, and I suggest you write it collaboratively using Office 365 or Google Docs. Please note that, even if a single report is uploaded, developers will be evaluated independently and can have different grades.
2. Include a header with course name, section, game name, the license of the report, and team composition. For each student, include name and section.
3. For each part of the lab, make sure to include the following:
   a) Part I: answers to numbered questions.

   b) Part II: the link of the shared repository. I’ll be able to check the history of your repository and see of everyone performed the described tasks. Please note: each team member should have a least one commit in the shared repository.
   c) Part III: the link to the GDD. If you are not writing it online, send me the current version in a separate PDF file. At this point, you should have the basic structure setup, and at least a review of the information sent with the pitch, now with the whole group input.
4. Save your report as PDF and upload it to Omnivox.

Good luck!