TP de projet de méthodologie avancée

Niveau 0

2020 - 2021

Ce niveau consiste à prendre en main différents outils dont vous aurez besoin pour le projet: la bibliothèque SDL2, le gestionnaire de version git, un makefile et un générateur de documentation (`doxygen`). **Attention**: pour faire ce TP, il est nécessaire d'avoir installé la bibliothèque SDL2 et l'outil `doxygen`.

1 Choix des binomes

Pour le début de la séance, indiquer sur le calepin "Formation des binomes" sur Arche la constitution de votre binome. **Attention:** les membres du binome doivent faire partie du même groupe de TP. Si vous n'arrivez pas à trouver de binome dans votre groupe, vous pouvez essayer d'en trouver un dans un autre groupe qui a le même créneau horaire que vous sur l'emploi du temps. Mais vous devez demander la permission aux chargés de TP correspondants. Si vous ne trouvez vraiment personne, indiquer votre nom dans la section dédiée du même calepin et contacter votre chargé de TP.

2 Préambule

Nous allons prendre en main la bibliothèque SDL 2. Il existe des documentations dont celles-ci:

- Documentation SDL2: `https://wiki.libsdl.org/FrontPage`
- Un tutoriel SDL 2 en français:
  `https://zestedesavoir.com/tutoriels/1014/utiliser-la-sdl-en-langage-c/`
- Pour l'installation sur votre machine, il vous faut :
  - la version développement de la bibliothèque SDL2
  - la prise en charge des polices TTF : libsdl2_ttf

Pour celles et ceux qui sont sur Windows, il existe, sur la page Arche du cours, des documents pour vous aider aux différentes installations. Toutes ces bibliothèques sont installées sur les machines linux de l'université. Vous n'avez donc pas à les installer lorsque vous êtes en présentiel et que vous utilisez ces machines.
Pour vous aider à faire le TP, vous avez aussi les supports et le code des séances des EI disponibles sur Arche.

1. Récupérer l'archive `niveau0.zip` (ressources) sur Arche qui contient:
   - les fichiers `sdl2-light.h` et `sdl2-light.c` correspondent à une sur-couche de SDL2 pour vous faciliter le travail

- le fichier `main.c` correspond à un programme qui affiche juste un fond et se termine lorsque l'on clique sur la croix de la fenêtre; le programme affiche aussi un message sur la sortie standard lorsque l'utilisateur appuie sur la touche 'D'
- l'image `background.bmp` correspond au fond de l'application.
- l'image `sprite.bmp` correspond à l'image d'un personnage.
- un fichier `Makefile` qui permet de compiler le programme automatiquement via la commande `make`; il existe une version Windows du Makefile (`Makefile_win`); si vous êtes sur Windows sans utiliser de machine virtuelle Linux, vous pouvez remplacer le fichier `Makefile` par `Makefile_win`.

A noter que le découpage du code pourrait être amélioré. Mais c'est l'objet de futurs sujets de TP.

2. Compiler le programme à l'aide du Makefile. Ne pas oublier d'ouvrir le fichier Makefile pour voir à quoi il ressemble.

3. À quoi servent les commandes `` `sdl2-config --cflags` `` et `` `sdl2-config --libs` `` ?

4. Exécuter le programme `spacecorridor`.

5. Générer la documentation `html` et `PDF` pour les fichiers `main.c` et `sdl2-light.c` avec `doxygen`, en tapant la commande `make doc` (voir dans le `Makefile` à quoi correspond cette cible).

6. Vérifier le résultat dans votre navigateur d'une part, et en visualisant le fichier `latex/refman.pdf` d'autre part.

3 Prise en main de la boucle de jeu

1. Ouvrir le fichier `main.c`, et retrouver les différentes étapes de la boucle de jeu, en vous aidant de la documentation.

2. Rajouter un champ `sprite` de type `SDL_Texture*` dans la structure `struct textures_s` qui contiendra les textures utiles au programme. Ce champ stockera la texture liée au personnage.

3. Dans la fonction `init_textures`, initialiser ce champ en chargeant l'image du personnage à l'aide de la fonction `load_image`.

4. Modifier la fonction `clean_textures` afin de nettoyer en mémoire la texture du personnage à la fin de l'exécution du programme.

5. Dans la fonction `refresh_graphics`, faire en sorte de placer le personnage pile au centre de l'écran de jeu à l'aide de la fonction `apply_texture`. Rappel : l'origine du repère cartésien de l'écran est en haut à gauche. L'axe des abscisses est orienté vers la droite, alors que l'axe des ordonnées est orienté vers le bas. Vous pouvez utiliser les constantes SCREEN_WIDTH, SCREEN_HEIGHT et SPRITE_SIZE qui donnent respectivement la largeur de l'écran (de jeu), la hauteur de l'écran, et la taille du personnage.

6. Tester votre programme. Ne pas oublier de mettre la documentation à jour et de la regénérer avec `doxygen`.

7. On souhaite maintenant ajouter les informations liées au personnage dans les données du monde. Ajouter dans la structure `struct world_s` la position (x,y) du personnage dans le repère de l'écran de jeu. Les deux champs correspondants sont des entiers.

8. Modifier l'initialisation des données (dans la fonction `init_data`) pour positionner initialement le personnage au centre de l'écran.

9. Modifier la fonction `refresh_graphics` pour placer le personnage de l'interface en fonction de la position du personnage dans les données du monde.

10. Tester votre programme. N'oubliez pas de mettre à jour votre documentation.

11. Nous souhaitons maintenant gérer les déplacements du personnage à l'aide des flèches du clavier. En appuyant sur une flèche, le personnage se déplacera d'un pas MOVING_STEP dans la direction correspondante à la flèche. Pour cela, modifier la fonction `handle_events`. Pour connaitre les constantes SDL liées aux différentes touches du clavier, aller sur `https://wiki.libsdl.org/SDL_Keycode`. N'hésitez pas à changer la valeur de MOVING_STEP pour que le personnage se déplace plus vite.

12. Tester votre programme.

13. Ajouter la prise en compte de l'appui sur la touche `Echap` pour quitter le jeu.

4 Prise en main de git

Pour le projet, vous allez utiliser l'application de gestion de projets `gitlab` hébergée à l'université, qui s'appuie sur l'outil `git`. Git est un gestionnaire de versions qui vous permet de stocker votre code source et d'en gérer les différentes versions. Il facilite le travail collaboratif. Lors de votre projet, vous devez obligatoirement l'utiliser. Vous devrez, au minimum, avoir une nouvelle version de votre code sur le serveur à chaque fin de séance. La notation du projet tiendra compte de ce que vous avez mis sur git.

1. Chacun des membres du binome doit se connecter à l'application
   `http://gitlab.univ-lorraine.fr`
   sur l'onglet LDAP avec son nom d'utilisateur/mot de passe de l'université.

   Travail pour le premier membre du binome (membre1)

2. La seconde étape va être de créer un projet git pour votre binome sur l'application `gitlab` hébergée à l'université. Pour cela, l'un des membres (membre1) de votre binome devra cliquer sur le "+" qui se trouve sur la barre de menu en haut de la page. Puis sélectionner "New project". La page de création de projet va alors apparaître.

3. Remplir le formulaire de création de projet de la manière suivante

   - le nom du projet (`Project name`) doit avoir la forme `methodo2021-X-tp-Y-nom1-nom2` où X est le nom de votre formation (ispi ou mi), Y est le numéro de votre groupe de TP (P1, P2, P3, D1, D2, D3, 4.1, 4.2); nom1 et nom2 correspondent aux noms de famille des membres du binome.
     Par exemple, si vous appartenez au groupe de TP D3 de la formation I-SPI et que les membres du binome s'appellent Dupont et Martin, le nom du projet sera
     `methodo2021-ispi-tp-D3-dupont-martin`. Si ce même binome appartient au groupe 4.1 de la formation M-I, alors le nom du projet sera `methodo2021-mi-tp-4.1-dupont-martin`.

- Sélectionner `Private` pour le niveau de visibilité (`Visibility Level`).
- Cocher la case `initialize repository with a README`.
- Cliquer sur le bouton `Create project`.

4. Vous allez maintenant ajouter votre collègue de binome (membre2) comme membre du projet, ainsi que le responsable du cours (Mathieu Constant – `Mathieu.Constant@univ-lorraine.fr`) et votre chargé de TP. Pour cela, aller sur la page du projet (cliquer sur `Projects`>`Your projects` sur la barre de menu du haut, puis cliquer sur votre projet). Ensuite, vous allez cliquer sur `Settings` > `Members` sur la barre de menu verticale à gauche.

5. Remplir le formulaire de la manière suivante:

   - Dans le champs `GitLab member or Email address`, rentrer le nom ou l'adresse mail de la personne que vous souhaitez ajouter au projet.
   - Dans le champs `Choose a role permission`, sélectionner `Maintainer`.
   - Puis, cliquer sur le bouton `Invite`.
   - Rappel: vous devez ajouter votre collègue de binome, ainsi que le responsable du cours et votre chargé de TP.
   - **Attention**: pour que vous puissiez trouver votre collègue de binome lors de l'ajout de membres, il est nécessaire que celui-ci se soit connecté au moins une fois à l'application.

Travail pour le deuxième membre du binome (membre 2)

6. L'autre membre (membre2) du binome doit maintenant être connecté à l'application `http://gitlab.univ-lorraine.fr`
   Cliquer sur le lien `Projects` > `Your projects` en haut à gauche. Cliquer ensuite sur votre projet. Vous constaterez que le projet contient uniquement le fichier README.

7. Cliquer sur le bouton `clone`, puis copier le contenu du champ associé (ce champ contient l'url de votre projet).

8. Le membre2 doit maintenant ouvrir un terminal, se placer dans son répertoire de travail. Lancer la commande `git clone CHEMIN` où CHEMIN correspond au contenu que vous avez copié sur `gitlab`. Cette commande va vous permettre de récupérer votre dépôt contenant uniquement un fichier README pour l'instant.

9. Aller dans le répertoire créé.

10. Créer un répertoire `tests`. Déplacez-y vos fichiers .h et .c, Makefile, ainsi que les images de l'exercice précédent.

11. La commande `git add chemin_fichier` permet d'indiquer que le fichier `chemin_fichier` soit suivi par le gestionnaire de version. Par exemple, si vous tapez la commande `git add tests/main.c`, le fichier `main.c` du répertoire `tests` sera désormais suivi par git. Faites en sorte que tous les fichiers que vous avez dans `tests` (.c, .h, Makefile et images) soient suivis.

12. La commande `git commit -a -m "mon message"` permet de mettre à jour les fichiers suivis dans le dépôt local à votre machine. L'option obligatoire `-m` permet d'écrire un message décrivant brièvement les modifications réalisées par rapport à la précédente version (dans la commande donnée, le message est "mon message"). Créer maintenant une nouvelle version de votre code en indiquant le message "version initiale".

13. Pour envoyer cette nouvelle version au serveur `gitlab` de l'université, vous devez utiliser la commande `git push`. Tester. Pour vérifier que cela a marché, aller sur la page de votre projet sur l'application `gitlab` et vérifier que les fichiers sont bien présents.

Travail pour le membre 1 du binome

14. Le membre1 du binome doit maintenant se placer dans le répertoire de son choix sur sa machine.

15. Faire un clone de votre dépôt git (commande `git clone` de tout à l'heure).

16. Modifier maintenant le fichier `main.c` en ajoutant des commentaires pertinents.

17. Créer une nouvelle version (commande `git commit` de tout à l'heure avec le message approprié) puis faire un envoi sur le serveur avec la commande `git push`.

18. Vérifier que c'est bien la bonne version qui se trouve sur l'application `gitlab`.

Travail pour le membre 2 du binome

19. Pour mettre à jour la bonne version du code sur sa machine, le membre2 du binome doit taper la commande `git pull` depuis le répertoire avec le code.

20. Vérifier que le fichier `main.c` est maintenant bien dans la bonne version.

Résumé du protocole standard d'utilisation de git lors d'une séance de travail

1) Modifications des fichiers du projet sur la machine de travail (phase de développement)
2) Marquage des fichiers à mettre à jour dans le dépôt **local** via `git add ...`
3) Mise à jour du dépôt **local** via `git commit -m "..."`
4) Récupération du dépôt **distant** pour fusion éventuelle via `git pull`
5) Modifications éventuelles des fichiers du projet pour résoudre les conflits lors de la fusion des deux dépôt (local et distant) :
   → Choix à faire dans les fichiers concernés entre les parties délimitées par :

```
<<<<<<<
version locale
=======
version distante
>>>>>>>
```

6) Mise à jour du dépôt **local** via `git commit ...`
7) Vérification qu'il n'y a pas eu de modifications du dépôt **distant** entre temps via `git pull`
8) S'il y a encore une fusion avec conflits, on recommence à l'étape 5)
9) Sinon, mise à jour du dépôt **distant** via `git push`

A noter qu'il est possible de fusionner les deux étapes `git add` et `git commit` en utilisant `git commit -a` comme montré dans l'énoncé. Attention: cela ne fonctionne que si tous les fichiers que vous avez modifiés sont déjà suivis par git (vous avez au moins fait une fois un `git add` sur ces fichiers). Si vous vous voulez inclure dans votre dépôt local un nouveau fichier, vous devez utiliser la commande `git add`.

Travail pour les deux membres du binome

21. Maintenant, en parallèle, sur leurs propres machines, les deux membres du binome doivent rajouter un nouveau commentaire dans le fichier `main.c`, chacun à deux endroits différents du code. Créer une nouvelle version locale chacun sur votre machine via un `git commit`.

22. Le membre 1 doit d'abord faire un envoi sur le serveur via un `git push`.

23. Que se passe-t-il si le membre 2 fait un envoi de sa version sur le serveur via un `git push`?

24. Pour résoudre le conflit de versions, le membre 2 doit d'abord récupérer la version la plus récente sur le serveur via la commande `git pull`. L'outil va ensuite effectuer la fusion des deux versions. Attention: la validation de cette opération s'effectue en tapant `:wq` (si le texte apparaît sur l'éditeur vim du terminal). N'hésitez pas à demander de l'aide à votre chargé de TP si vous n'y arrivez pas. Cette opération fusionne les deux versions et en fait une nouvelle.

25. Terminer en faisant un push sur le serveur pour mettre cette version sur le `gitlab`.

26. Refaire la même séquence d'opérations en inversant les rôles, de sorte que le membre 1 ait à résoudre le conflit de versions.

Vous connaissez la base des bases pour utiliser git. Attention: il peut se trouver des situations où les conflits ne peuvent pas être résolus automatiquement (la fusion n'a pas marché). Vous serez amené à les gérer manuellement. Il existe d'autres procédés pour régler les conflits avec git. Pour en savoir plus, allez sur `https://git-scm.com/book/fr/v1/`. Si vous êtes bloqués, n'hésitez pas à faire appel à votre chargé de TP.

Bravo ! Vous avez terminé le niveau 0 !