Squirrel Battle

Yohann D'ANELLO, Mathilde DEPRES, Nicolas MARGULIES, Char

janv. 25, 2021

Développer

CHAPITRE 1

Installation d'un environnement de développement

Il est toujours préférable de travailler dans un environnement Python isolé du reste de son instalation.

1. **Installation des dépendances de la distribution.** Vous devez déjà installer Python et le module qui permet de créer des environnements virtuels. On donne ci-dessous l'exemple pour une distribution basée sur Debian, mais vous pouvez facilement adapter pour ArchLinux ou autre.

```
$ sudo apt update
$ sudo apt install --no-install-recommends -y python3-setuptools python3-venv python3-
↪dev gettext git
```

2. **Clonage du dépot** là où vous voulez :

```
$ git clone git@gitlab.crans.org:ynerant/squirrel-battle.git && cd squirrel-battle
```

3. **Création d'un environment de travail Python décorrélé du système.** On n'utilise pas *–system-site-packages* ici pour ne pas avoir des clashs de versions de modules avec le système.

```
$ python3 -m venv env
$ source env/bin/activate  # entrer dans l'environnement
(env) $ pip3 install -r requirements.txt
(env) $ deactivate  # sortir de l'environnement
```

4. **Compilation des messages de traduction.**

```
(env) $ python3 main.py --compilemessages
```

Le lancement du jeu se fait en lançant la commande `python3 main.py`.

CHAPITRE 2

Exécution des tests

`tox` est un outil permettant de configurer l'exécution des tests. Ainsi, après installation de tox dans votre environnement virtuel via `pip install tox`, il vous suffit d'exécuter `tox -e py3` pour lancer les tests et `tox -e linters` pour vérifier la syntaxe du code.

2.1 Tests unitaires

Les tests sont gérés par `pytest` dans le module `squirrelbattle.tests`. Le module `pytest-cov` permet de mesurer la couverture du code. Pour lancer les tests, il suffit de lancer `tox -e py3` ou de manière équivalente `pytest --cov=squirrelbattle/ --cov-report=term-missing squirrelbattle/`

L'intégration continue lance les tests pour les versions de Python de 3.6 à 3.10, sur une distribution Alpine Linux.

Chaque partie du code est testée unitairement, pour obtenir une couverture maximale et assurer un bon fonctionnement. En particulier, le jeu est lancé en commençant sur une carte déterministe (non générée aléatoirement) chargée depuis `assets/example_map.txt`, sur laquelle sont placés des ennemis et objets avec lesquels le joueur doit interagir. On vérifie qu'à chaque touche appuyée, il se passe le bon comportement. Le comportement des différents menus est également testé.

L'environnement de test ne disposant pas a priori d'un terminal, le jeu est conçu pour fonctionner sans support graphique, avec un terminal fictif où les primitives de curses sont implémentées pour ne rien faire. On ne peut alors pas s'assurer du bon fonctionnement de curses.

De plus, une très fine partie du code est ignorée lors de la couverture, ce qui correspond à la phase d'initialisation du terminal et à la boucle infinie qui reçoit les touches de l'utilisateur, qu'il est alors impossible de tester unitairement.

2.2 Analyseur syntaxique

`flake8` est utilisé en guise d'analyseur syntaxique. Il vérifie si le code est bien formatté afin d'assurer une certaine lisibilité. En particulier, il vérifie l'indentation, si chaque variable est bien utilisée, s'il n'y a pas d'import inutile et s'ils sont dans l'ordre lexicographique, si chaque ligne fait au plus 80 caractères et si la signature de chaque fonction est bien spécifiée.

Pour lancer l'analyse, `tox -e linters` suffit. L'intégration continue effectue cette analyse à chaque commit.

CHAPITRE 3

Gestion de l'affichage

3.1 Affichage des menus

Les menus sont affichés dans une boîte. On peut naviguer dedans avec les flèches haut et bas, et valider avec la touche entrée. Il est également possible d'intéragir avec la souris.

Il y a plusieurs menus dans le jeu :

- **Le menu principal**, qui s'affiche au lancement du jeu.
- **Le menu des paramètres** : si on sélectionne un choix de touche et qu'on appuie sur entrée, on peut ensuite appuyer sur une touche pour remplacer la touche utilisée.
- **Le menu des crédits** : ce menu fonctionne avec la souris. En cliquant on affiche une image.
- **Le menu d'inventaire** : dans l'inventaire, on peut utiliser les touches pour utiliser un item ou l'équiper...
- **Le menu de vente** : on peut utiliser les touches gauche et droite pour switcher entre l'inventaire du joueur et celui du marchand.
- **Menu des warnings** : Pas vraiment un menu, mais affiche juste un message dans une petite boite pour prévenir le joueur que quelque chose ne va pas.

3.2 Affichage de la carte

La carte s'affiche dans la partie en haut à gauche de l'écran, sur la plus grande partie de l'écran. On affiche les tuiles une par une. Selon le pack de textures utilisé, les tuiles prennent un ou deux caractères de large. Selon la visibilité de la case en fonction de la position du joueur, la couleur de la case sera plus ou moins sombre, voire masquée si le joueur n'a jamais vu la case. Les entités sont ensuite affichées, si elles sont dans le champ de vision du joueur.

La carte est actualisée lorsque cela est nécessaire, à chaque tick de jeu.

L'afffichage de la carte suit les déplacements du joueur, dans le sens où la caméra est toujours centrée sur lui. La carte prend 4/5 de l'affichage aussi bien en largeur qu'en hauteur.

3.3 Affichage des statistiques

Les statistiques du joueur sont affichées en haut à droite de l'écran et séparées du reste de l'affichage par une barre verticale, occupant 1/5 de la place horizontale.

Les informations affichées sont :
- **LVL** - le niveau du joueur
- **EXP** - la quantité d'expérience que le joueur a gagné et combien il lui en faut avant le prochain niveau.
- **HP** - la quantité de vie que le joueur a actuellement et combien il peut en avoir au maximum.
- **STR** - la force du joueur.
- **INT** - l'intelligence du joueur.
- **CHR** - le charisme du joueur.
- **DEX** - la dextérité du joueur.
- **CON** - la constitution du joueur.
- **CRI** - le pourcentage de chance de coup critique.
- **Inventory** - le contenu de l'inventaire du joueur.
- **Equipped main** - l'objet équipé dans la main principale.
- **Equipped secondary** - l'objet équipé dans la main secondaire.
- **Equipped armor** - le plastron porté par le joueur.
- **Equipped helmet** - le casque porté par le joueur.
- **Hazel** - le nombre d'"Hazel que le joueur possède.
- **Vous êtes mort** - Éventuellement, si le joueur est mort.

Si le joueur possède un monocle, alors les statistiques d'une entité proche sont également affichées dessous.

Des aides de jeu peuvent enfin être affichées en bas, telles que la touche sur laquelle il faut appuyer.

3.4 Affichage de l'historique

L'historique des actions est affiché en bas de l'écran. À chaque action d'une entité, comme frapper quelqu'un, ou lorsque le joueur parle à une entité, cela s'affiche dans l'historique.

Il est affiché sur l'écran de jeu, en bas à gauche, occupant 1/5 de la hauteur et 4/5 de la largeur.

L'intégralité de l'affichage du jeu est géré grâce à la bibliothèque native de Python curses.

3.5 Initialisation du terminal

Au lancement du jeu, le terminal est initialisé, les caractères spéciaux sont traduits en abstraction curses, les caractères tapés par l'utilisateur ne sont plus affichés sur le terminal, on envoie les touches tapées à curses en permanence sans avoir à taper sur Entrée, le curseur est rendu invisible, on active le support des couleurs et enfin on déclare vouloir attraper tous les clics de souris. Tout cela est fait dans un contexte Python, qui permet d'effectuer les opérations inverses lors de la fermeture du programme, même en cas de crash, afin de retrouver un terminal utilisable.

3.6 Pads

Chaque morceau d'affichage est géré dans un *pad*. Un pad est un objet défini par curses, représentant une sous-fenêtre, qui a l'avantage d'être un peu plus flexible qu'une simple fenêtre. Un pad peut en effet dépasser de l'écran, et on peut choisir où placer le coin avant gauche et quelle partie du pad il faut dessiner sur la fenêtre.

Ce projet implémente une couche d'abstraction supplémentaire, afin d'avoir des objets plus utilisables. Chaque partie d'affichage est réprésentée dans une classé étendant `Display`. Chaque `Display` contient un (ou plusieurs) pads, et propose une surcouche de certaines fonctions de curses.

L'affichage de texte par exemple sur un pad est plus sécurisée que celle proposée par curses. Le comportement par défaut est d'afficher un message à partir d'une position donnée avec des attributs (gras, couleur, ...) donnés sous forme numérique. Cette implémentation permet de passer les attributs voulus sous forme de paramètres booléens, de choisir la couleur de front/de fond sans définir de paire curses, mais surtout de tronquer le texte à la place disponible, afin de ne pas avoir à se soucier d'avoir un message trop grand et éviter des crashs non désirés.

Les `Display` sont gérés par un `DisplayManager`. C'est lui qui décide, en fonction de l'état actuel du jeu, d'afficher les bons `Display` aux bons endroits et de les redimensionner correctement en fonction de la taille du terminal. C'est aussi lui qui propage l'information de modifier les attributs d'un `Display`, si par exemple l'inventaire du joueur a été mis à jour.

Il s'occupe enfin de tout redimensionner si jamais le terminal a changé de taille, après une intervention de l'utilisateur.

3.7 Interactions avec la souris

Le jeu attrape les clics de souris. C'est le `DisplayManager`, connaissant l'état du jeu et ce qui est affiché à quel endroit, qui va chercher sur quel `Display` on a cliqué. L'information est propagée au bon `Display`, en adaptant les coordonnées.

Tout `Display` qui contient un menu procède de la même façon pour propager l'information au bon menu.

CHAPITRE 4

Traduction

Le jeu Squirrel Battle est entièrement traduit en anglais, en français et en allement. La langue se choisit dans les paramètres.

4.1 Utitisation

Les traductions sont gérées grâce au module natif `gettext`. Le module `squirrelbattle.translations` s'occupe d'installer les traductions, et de donner les chaînes traduites.

Pour choisir la langue, il faut appeler `Translator.setlocale(language: str)`, où `language` correspond au code à 2 lettres de la langue.

Enfin, le module expose une fonction `gettext(str) -> str` qui permet de traduire les chaînes.

Il est courant et recommandé d'importer cette fonction sous l'alias _, afin de limiter la verbositer et de permettre de rendre facilement une chaîne traduisible.

```
from squirrelbattle.translations import gettext as _, Translator

Translator.setlocale("fr")
print(_("I am a translatable string"))
print("I am not translatable")
```

Si les traductions sont bien faites (voir ci-dessous), cela donnera :

```
Je suis une chaîne traduisible
I am not translatable
```

À noter que si la chaîne n'est pas traduite, alors par défaut on renvoie la chaîne elle-même.

4.2 Extraction des chaînes à traduire

L'appel à `gettext` ne fait pas que traduire les chaînes : il est possible également d'extraire toutes les chaînes à traduire.

Il est nécessaire d'installer le paquet Linux `gettext` pour cela.

L'utilitaire `xgettext` s'occupe de cette extraction. Il s'utilise de la façon suivante :

```
xgettext --from-code utf-8 -o output_file.po source_1.py ... source_n.py
```

Afin de ne pas avoir à sélectionner manuellement chaque fichier, il est possible d'appeler directement `python3 main.py --makemessages`. Cela a pour effet d'exécuter pour chaque langue `<LANG>` :

```
find squirrelbattle -iname '*.py' | xargs xgettext --from-code utf-8
                  --add-comments
                  --package-name=squirrelbattle
                  --package-version=23.14
                  "--copyright-holder=ÿnérant, eichhornchen, nicomarg, charlse"
                  --msgid-bugs-address=squirrel-battle@crans.org
                  -o squirrelbattle/locale/<LANG>/LC_MESSAGES/squirrelbattle.po
```

Les fichiers de traductions se trouvent alors dans `squirrelbattle/locale/<LANG>/LC_MESSAGES/squirrelbattle.po`.

4.3 Traduire les chaînes

Après extraction des chaînes, les chaînes à traduire se trouvent dans `squirrelbattle/locale/<LANG>/LC_MESSAGES/squirrelbattle.po`, comme indiqué ci-dessus.

Ce fichier peut-être édité avec un utilitaire tel que `poedit`, sur l'interface Web sur https://translate.ynerant.fr/squirrel-battle/squirrel-battle, mais surtout manuellement avec un éditeur de texte.

Dans ce fichier, on obtient pour chaque chaîne à traduire un paragraphe de la forme :

```
#: main.py:4
msgid "I am a translatable string"
msgstr "Je suis une chaîne traduisible"
```

Il sufift de remplir les champs `msgstr`.

4.4 Compilation des chaînes

Pour gagner en efficacité, les chaînes sont compilées dans un fichier avec l'extension `.mo`. Ce sont ces fichiers qui sont lus par le module de traduction.

Pour compiler les traductions, c'est l'utilitaire `msgfmt` fourni toujours par le paquet Linux `gettext` que nous utilisons. Il s'utilise assez simplement :

```
msgfmt po_file.po -o mo_file.mo
```

À nouveau, il est possible de compiler automatiquement les messages en exécutant `python3 main.py --compilemessages`.

Avertissement : On ne partagera pas dans le dépôt Git les fichiers compilé. En développement, on compilera soi-même les messages, et en production, la construction des paquets se charge de compiler automatiquement les traductions.

CHAPITRE 5

Déploiement du projet

À chaque nouvelle version du projet, il est compilé et déployé dans PyPI, dans l'AUR et un paquet Debian est créé, voir la page d'installation.

5.1 PyPI

5.1.1 Définition du paquet

La documentation sur le packaging dans PyPI est disponible ici.

Le fichier setup.py contient l'ensemble des instructions d'installation du paquet ainsi que des détails à fournir à PyPI :

```
#!/usr/bin/env python3
import os

from setuptools import find_packages, setup

with open("README.md", "r") as f:
    long_description = f.read()

# Compile messages
for language in ["de", "es", "fr"]:
    args = ["msgfmt", "--check-format",
            "-o", f"squirrelbattle/locale/{language}/LC_MESSAGES"
                  "/squirrelbattle.mo",
            f"squirrelbattle/locale/{language}/LC_MESSAGES"
            "/squirrelbattle.po"]
    print(f"Compiling {language} messages...")
    subprocess.Popen(args)

setup(
    name="squirrel-battle",
    version="23.14",
```

(suite sur la page suivante)

(suite de la page précédente)

```
    author="ÿnérant, eichhornchen, nicomarg, charlse",
    author_email="squirrel-battle@crans.org",
    description="Watch out for squirrel's knives!",
    long_description=long_description,
    long_description_content_type="text/markdown",
    url="https://gitlab.crans.org/ynerant/squirrel-battle",
    packages=find_packages(),
    license='GPLv3',
    classifiers=[
        "Development Status :: 4 - Beta",
        "Environment :: Console :: Curses",
        "License :: OSI Approved :: GNU General Public License v3 (GPLv3)",
        "Natural Language :: French",
        "Operating System :: OS Independent",
        "Programming Language :: Python :: 3",
        "Programming Language :: Python :: 3.6",
        "Programming Language :: Python :: 3.7",
        "Programming Language :: Python :: 3.8",
        "Programming Language :: Python :: 3.9",
        "Topic :: Games/Entertainment",
    ],
    python_requires='>=3.6',
    include_package_data=True,
    package_data={"squirrelbattle": ["assets/*", "locale/*/*/*.mo"]},
    entry_points={
        "console_scripts": [
            "squirrel-battle = squirrelbattle.bootstrap:Bootstrap.run_game",
        ]
    }
)
```

Ce fichier contient le nom du paquet, sa version, l'auteur et son contact, sa description en une ligne et sa description longue, le lien d'accueil du projet, sa licence, ses classificateurs et son exécutable.

Il commence tout d'abord par compiler les fichiers de traduction.

Le paramètre `entry_points` définit un exécutable nommé `squirrel-battle`, qui permet de lancer le jeu.

5.1.2 Installation locale du paquet

L'installation du paquet localement dans son environnement Python (virtuel ou non) peut se faire en exécutant `pip install -e .`.

5.1.3 Génération des binaires

Les paquets `setuptools` (`python3-setuptools` pour APT, `python-setuptools` pour pacman) et `wheel` (`python3-wheel` pour APT, `python-wheel` pour pacman) sont nécessaires. Une fois installés, il faut appeler la commande :

```
python3 setup.py sdist bdist_wheel
```

Une fois cela fait, le dossier `dist/` contiendra les archives à transmettre à PyPI.

5.1.4 Publier sur PyPI

Il faut avant tout avoir un compte sur PyPI. Dans votre compte PyPI, il faut générer un jeton d'accès API.

Dans le fichier `.pypirc` dans le répertoire principal de l'utilisateur, il faut ajouter le jeton d'accès :

```
[pypi]
  username = __token__
  password = pypi-my-pypi-api-access-token
```

Cela permet de s'authentifier directement par ce jeton.

Ensuite, il faut installer `twine`, qui permet de publier sur PyPI.

Il suffit ensuite d'appeler :

```
twine upload dist/*
```

pour envoyer le paquet sur PyPI.

Note : À des fins de tests, il est possible d'utiliser le dépôt https://test.pypi.org. Les différences sont au niveau de l'authentification, où il faut l'en-tête `[testpypi]` dans le `.pypirc`, et il faut envoyer le paquet avec `twine upload --repository testpypi dist/`.

5.2 Publier dans l'AUR

5.2.1 Fonctionnement du packaging

Deux paquets sont publiés dans l'AUR (Arch User Repository) :

- python-squirrel-battle
- python-squirrel-battle-git

Le packaging dans Arch Linux se fait en commitant un fichier `PKGBUILD` dans le dépôt à l'adresse `ssh://aur@aur.archlinux.org/packagename.git`, en remplaçant `packagename` par le nom du paquet.

Le second paquet compile directement le jeu à partir de la branche `master` du dépôt Git. Le fichier `PKGBUILD` dispose de cette structure :

```
# Maintainer: Yohann D'ANELLO <squirrel-battle@crans.org>

pkgbase=squirrel-battle
pkgname=python-squirrel-battle-git
pkgver=23.14
pkgrel=1
pkgdesc="Watch out for squirrel's knives!"
arch=('any')
url="https://gitlab.crans.org/ynerant/squirrel-battle"
license=('GPLv3')
depends=('python')
makedepends=('gettext' 'python-setuptools')
depends=('noto-fonts-emoji')
checkdepends=('python-tox')
ssource=("git+https://gitlab.crans.org/ynerant/squirrel-battle.git")
sha256sums=("SKIP")

```

(suite sur la page suivante)

(suite de la page précédente)

```
pkgver() {
  cd pkgbase
  git describe --long --tags | sed -r 's/^v//;s/([^-]*-g)/r\1/;s/-/./g'
}
build() {
  cd $pkgbase
  python setup.py build
}

check() {
  cd $pkgbase
  tox -e py3
  tox -e linters
}

package() {
  cd $pkgbase
  python setup.py install --skip-build \
                          --optimize=1 \
                          --root="${pkgdir}"
  install -vDm 644 README.md \
    -t "${pkgdir}/usr/share/doc/${pkgname}"
  install -vDm 644 LICENSE -t "${pkgdir}/usr/share/licenses/${pkgname}"
}
```

Ces instructions permettent de cloner le dépôt, l'installer et exécuter des tests, en plus de définir les attributs du paquet.

Le fichier `PKGBUILD` du paquet `python-squirrel-battle`, synchronisé avec les releases, est plus ou moins similaire :

```
# Maintainer: Yohann D'ANELLO <squirrel-battle@crans.org>

pkgbase=squirrel-battle
pkgname=python-squirrel-battle
pkgver=23.14
pkgrel=1
pkgdesc="Watch out for squirrel's knives!"
arch=('any')
url="https://gitlab.crans.org/ynerant/squirrel-battle"
license=('GPLv3')
depends=('python')
makedepends=('gettext' 'python-setuptools')
depends=('noto-fonts-emoji')
checkdepends=('python-tox')
source=("https://gitlab.crans.org/ynerant/squirrel-battle/-/archive/v23.14/$pkgbase-v
↪$pkgver.tar.gz")
sha256sums=("6090534d598c0b3a8f5acdb553c12908ba8107d62d08e17747d1dbb397bddef0")

build() {
  cd $pkgbase-v$pkgver
  python setup.py build
}

check() {
  cd $pkgbase-v$pkgver
  tox -e py3
  tox -e linters
```

(suite sur la page suivante)

(suite de la page précédente)

```
}

package() {
  cd $pkgbase-v$pkgver
  python setup.py install --skip-build \
                          --optimize=1 \
                          --root="${pkgdir}"
  install -vDm 644 README.md \
    -t "${pkgdir}/usr/share/doc/${pkgname}"
  install -vDm 644 LICENSE -t "${pkgdir}/usr/share/licenses/${pkgname}"
}
```

Il se contente ici de télécharger l'archive de la dernière release, et de travailler dessus.

5.2.2 Mettre à jour

Pour mettre à jour le dépôt, une fois les dépôts `ssh://aur@aur.archlinux.org/python-squirrel-battle.git` et `ssh://aur@aur.archlinux.org/python-squirrel-battle-git.git` clonés, il suffit de mettre à jour le paramètre `pkgver` pour la bonne version, de régénérer le fichier `.SRCINFO` en faisant `makepkg --printsrcinfo > .SRCINFO`, puis de committer/pousser.

5.3 Construction du paquet Debian

5.3.1 Structure du paquet

L'ensemble des instructions pour construire le paquet Debian est situé dans le dossier `debian/`.

Le fichier `changelog` est à modifier à chaque nouvelle version, le fichier `compat` contient la version minimale de Debian requise (`10` pour Debian Buster), le fichier `copyright` contient la liste des fichiers distribués sous quelle licence (ici GPLv3), le fichier `control` contient les informations du paquet, le fichier `install` les fichiers de configuration à installer (ici le fix de l'affichage de l'écureuil), et enfin le fichier `rules` l'ensemble des instructions à exécuter pour installer.

Le paquet `fonts-noto-color-emoji` est en dépendance pour le bon affichage des émojis.

5.3.2 Mettre à jour le paquet

Pour changer la version du paquet, il faut ajouter des lignes dans le fichier `changelog`.

5.3.3 Construire le paquet

Il faut partir d'une installation de Debian.

D'abord on installe les paquets nécessaires :

```
apt update
apt --no-install-recommends install build-essential debmake dh-python debhelper␣
↪gettext python3-all python3-setuptools
```

On peut ensuite construire le paquet :

```
dpkg-buildpackage
mkdir build && cp ../*.deb build/
```

Le paquet sera installé dans `build/python3-squirrel-battle_23.14_all.deb`.

Le paquet Debian est construit par l'intégration continue Gitlab et ajouté à chaque release.

CHAPITRE 6

Documentation

La documentation est gérée grâce à Sphinx. Le thème est le thème officiel de ReadTheDocs `sphinx-rtd-theme`.

6.1 Générer localement la documentation

On commence par se rendre au bon endroit et installer les bonnes dépendances :

```
cd docs
pip install -r requirements.txt
```

La documentation se génère à partir d'appels à `make`, selon le type de documentation voulue.

Par exemple, `make html` construit la documentation web, `make latexpdf` construit un livre PDF avec cette documentation.

6.2 Documentation externe

À chaque commit, un webhook est envoyé à readthedocs.io, qui construit tout seul la documentation Sphinx, la publiant à l'adresse squirrel-battle.readthedocs.io.

De plus, les documentations sont sauvegardées à chaque release taguée.

CHAPITRE 7

Installation client

7.1 Installation

Différents paquets sont déployés, dans PyPI pour tout système utilisant Python, un paquet Debian et un paquet Arch Linux.

7.1.1 Depuis PIP

Le projet *Squirrel Battle* est déployé dans PyPI. Il suffit d'installer Squirrel Battle en exécutant :

```
pip install --user squirrel-battle
```

Les mises à jour s'obtiennent également via PIP en exécutant :

```
pip install --user --upgrade squirrel-battle
```

Le jeu peut se lancer ensuite en exécutant la commande `squirrel-battle`.

Toutefois, le paquet PyPI n'inclut pas les polices d'émojis. Il est recommandé d'installer des polices telles que `noto-fonts-emoji` afin de prendre en charge les émojis dans votre terminal.

7.1.2 Sur Arch Linux

Deux paquets sont publiés dans l'AUR (Arch User Repository) :

— python-squirrel-battle
— python-squirrel-battle-git

Le premier paquet est mis à jour à chaque nouvelle version déployée, le second est utile pour des fins de développement et est en permanence à jour avec la branche `master` du Git.

Les deux ne sont pas présents dans les dépôts officiels de Arch Linux, mais vous pouvez les récupérer avec un outil tel que yay.

Les paquets incluent la dépendance `noto-fonts-emoji`, qui permet d'afficher les émojis dans le terminal.

Le jeu peut être ensuite lancé via la commande `squirrel-battle`.

7.1.3 Sur Ubuntu/Debian

Un paquet est généré par l'intégration continue de Gitlab à chaque commit. Ils sont également attachés à chaque nouvelle release.

Il dépend du paquet `fonts-noto-color-emoji`, permettant d'afficher les émojis dans le terminal. Il peut être installé via APT.

Pour installer ce paquet, il suffit de le télécharger et d'appeler `dpkg` :

```
dpkg -i python3-squirrelbattle_23.14_all.deb
```

Ce paquet inclut un patch pour afficher les émojis écureuil correctement.

Après cela, le jeu peut être lancé grâce à la commande `squirrel-battle`.

CHAPITRE 8

Règles du jeu

Dans *Squirrel Game*, le joueur incarne un écureuil coincé dans un donjon, prêt à tout pour s'en sortir. Sa vision de rongeur lui permet d'observer l'intégralité de la carte, et à l'aide d'"objets, il va pouvoir affronter les monstres présents dans le donjon et gagner en expérience et en force.

Le jeu fonctionne par étage. À chaque étage, différents monstres sont présents, et à l'aide d'objets, il pourra progresser dans le donjon et descendre de plus en plus bas.

En tuant des ennemis, ce qu'il parvient à faire en fonçant directement sur eux ayant mangé trop de noisettes (ou étant armé d'un couteau), l'écureuil va pouvoir gagner en expérience et au fur et à mesure qu'il monte de niveau, a force augmentera.

Arriverez-vous à sauver ce malheureux petit écureuil perdu ?

Bon courage sachant que le jeu est sans fin ...

CHAPITRE 9

Carte

Dans Squirrel game, le joueur se déplace dans un donjon, constitué de plusieurs cartes. Pour le moment, le jeu se déroule sur une unique carte pré-définie, non générée aléatoirement.

Une carte est un rectangle composé de *tuiles*.

La carte est chargée depuis sa représentation ASCII dans un fichier texte.

Au lancement du jeu, une quantité aléatoire d'"entités sont générées et placées aléatoirement sur la carte.

9.1 Tuiles

Une tuile représente une case du jeu, avec ses différentes propriétés physiques. On compte actuellement 3 types de tuiles :

9.1.1 Vide

Le vide est représenté par un espace vide quelque que soit le pack de textures utilisé. Cette tuile n'est utilisée que pour délimiter les bords de la carte, aucune entité ne peut se trouver sur cette tuile.

9.1.2 Sol

Le sol représente les emplacements où les entités peuvent se déplacer librement. Il est représenté par un point `.` dans le pack de textures ASCII et par deux caractères rectangulaires blancs dans le pack de textures écureuil.

9.1.3 Mur

Les murs délimitent les salles du donjon. Personne ne peut les traverser. Ils sont représentés par un dièse `#` dans le pack de textures ASCII et par une brique carrée dans le pack de textures écureuil.

9.1.4 Échelles

Les échelles sont les débuts et fin de niveau. Elles permettent de changer d'étage en appuyant sur une touche. Elles sont représentées par un `H` dans le pack de textures ASCII et par un émoji échelle dans le pack de textures écureuil.

Lorsqu'on est sur l'échelle du début de niveau, appuyer sur < permet de monter d'un étage (revenir au niveau précédent). Lorsqu'on est sur l'échelle de fin de niveau, on génère une nouvelle carte si ce n'est pas déjà fait avec des monstres plus forts, et on place le joueur sur cette nouvelle carte.

CHAPITRE 10

Entités

10.1 Joueur

Le joueur est une entité attaquante, contrôlée par l'utilisateur humain.

Il est représenté dans le pack de textures ASCII par le caractère `@`, et dans le pack de textures écureuil par le fameux émoji écureuil .

En plus des attributs d'une entité attaquante, le joueur dispose des atrributs supplémentaires :

— `current_xp: int`
Correspond à l'expérience accumulée par le joueur depuis le dernier niveau obtenu.

— `max_xp: int`
Expérience requise au joueur pour changer de niveau. Vaut 10 fois le niveau actuel.

— `inventory: List[Item]`
Contient l'ensemble des objets détenus par le joueur.

10.1.1 Déplacement

Selon les paramètres, il est possible de bouger le joueur dans les 4 directions en appuyant sur `z`, `q`, `s`, `d` ou sur les flèches directionnelles. (ou sur d'autres touches selon ce qui est écrit dans le menu des paramètres)

Le joueur peut aussi ne rien faire pendant son tour, il suffit d'appuyer sur la touche d'attente (`w` de base).

Le joueur se retrouvera bloqué s'il avance contre un mur. Si il avance sur un objet, alors il prend l'"objet et avance sur la case.

S'il rencontre une autre entité attaquante, alors il frappe l'entité en infligeant autant de dégâts qu'il n'a de force. À chaque fois qu'une entité est tuée, le joueur gagne aléatoirement entre 3 et 7 points d'expérience.

Outre se déplacer et attaquer, le joueur peut utiliser la touche pour danser (`y` de base) durant son tour et danser. Selon son charisme, il a plus ou moins de chances de rendre confus tous les ennemis à distance moins de 3. Un ennemi confus ne peut pas attaquer.

10.1.2 Statistiques

Le joueur possède plusieurs statistiques :
- Niveau : son niveau, qui dépend de combien d'expérience il a accumulé
- Expérience : la quantité d'expérience accumulée par le joueur, qui dépend de combien d'entités il a tué.
- Force : indique combien de dommages le joueur inflige à ses ennemis
- Intelligence : joue sur l'effet des objets magiques, tels que les parchemins ou les batons
- Charisme : joue sur l'efficacité de la danse du joueur
- Dextérité : joue sur l'efficacité de l'"arc
- Constitution : joue sur la quantité de dégâts que le joueur prend lorsqu'un monstre le frappe
- Taux de critique : la chance (en pourcentage) que le joueur a de faire un coup critique

10.1.3 Expérience

À chaque monstre tué, le joueur gagne entre 3 et 7 points d'expérience aléatoirement. Lorsque le joueur atteint la quantité d'expérience requise pour monter de niveau, le joueur gagne un niveau, regagne toute sa vie, consomme son expérience et la nouvelle quantité d'expérience requise est 10 fois le niveau actuel. De plus, entre 5 et 10 fois le niveau actuel entités apparaissent aléatoirement sur la carte à la montée de niveau. Enfin, le joueur améliore ses statistiques en augmentant de niveau. Toutes les caractéristiques ne sont pas incrémentées à chaque niveau gagné.

10.2 Monstres

Chaque monstre est une entité attaquante, et hérite donc de ses attributs.

À chaque tick de jeu, chaque monstre se déplace d'une case, si possible. Si le monstre est loin du joueur, ce déplacement est fait aléatoirement. Sinon, si le monstre est à moins de 5 cases du joueur, alors il se dirige au plus vite sur le joueur pour le frapper selon l'algorithme de Dijkstra, et s'il est suffisamment proche frappe le joueur et lui fait autant de dégâts qu'il n'a de force.

On dénombre actuellement 5 types de monstres :

10.2.1 Hérisson

Son nom est fixé à *hedghog*. Il a par défaut une force à **3** et **10** points de vie.

Dans le pack de textures ASCII, il est représenté par le caractère `*`.

Dans le pack de textures écureuil, il est représenté par l'émoji .

10.2.2 Tigre

Son nom est fixé à *tiger*. Il a par défaut une force à **2** et **20** points de vie.

Dans le pack de textures ASCII, il est représenté par le caractère `n`.

Dans le pack de textures écureuil, il est représenté par l'émoji .

10.2.3 Lapin

Son nom est fixé à *rabbit*. Il a par défaut une force à **1** et **15** points de vie.

Il a une chance de coup critique de 30%.

Dans le pack de textures ASCII, il est représenté par le caractère `Y`.

Dans le pack de textures écureuil, il est représenté par l'émoji .

10.2.4 Nounours

Son nom est fixé à *teddy_bear*. Il n'a pas de force et **50** points de vie.

Dans le pack de textures ASCII, il est représenté par le caractère `8`.

Dans le pack de textures écureuil, il est représenté par l'émoji .

10.2.5 Pygargue

Son nom est fixé à *eagle*. Il a par défaut une force à **1000** et **5000** points de vie.

Il s'agit d'un boss difficilement tuable, qui apparait plus rarement que les autres monstres.

Dans le pack de textures ASCII, il est représenté par le caractère `µ`.

Dans le pack de textures écureuil, il est représenté par l'émoji .

10.3 Entités pacifiques

Chaque entité pacifique est en particulier une entité attaquante, et hérite donc de ses attributs, et peut alors être attaquée. Ils sont cependant non-hostiles.

Il est possible d'interagir avec ces entités. En s'approchant d'elles, en appuyant sur la touche `T` suivie de la direction où regarder, un échange débute.

Si l'on s'adresse à un marchand, on devrait voir à l'écran l'inventaire du joueur et l'inventaire du marchand. Les flèches haut et bas permettent de sélectionner un objet, les touches droite et gauche de passer d'un inventaire à l'autre, et la touche entrée valide l'action.

On dénombre actuellement 3 types d'entités pacifiques :

10.3.1 Tournesol

Son nom est fixé à *sunflower*. Il a par défaut une **15** points de vie.

Interagir avec un tournesol n'a pas de réel intérêt, si ce n'est déclencher le « pouvoir des fleurs ! ! » ou bien savoir que « le soleil est chaud aujourd'hui ».

Dans le pack de textures ASCII, il est représenté par le caractère `I`.

Dans le pack de textures écureuil, il est représenté par l'émoji .

10.3.2 Marchand

Son nom est fixé à *merchant*. Il a par défaut **5** points de vie.

En interagissant avec un marchand, il est possible de lui acheter et de lui vendre différents objets contre des Hazels, la monnaie du jeu. Les prix sont fixés :

- Anneau de coup critique : 15 Hazels
- Anneau d'expérience : 25 Hazels
- Arc : 22 Hazels
- Baguette de feu : 36 Hazels
- Bombe : 4 Hazels
- Bouclier : 16 Hazels
- Casque : 18 Hazels
- Coeur : 3 Hazels
- Épée : 20 Hazels
- Monocle : 10 Hazels
- Parchemin de dégâts : 18 Hazels
- Parchemin de faiblesse : 13 Hazels
- Plastron : 30 Hazels
- Potion d'arrachage de corps : 14 Hazels
- Règle : 2 Hazels

Le marchand commence avec 75 Hazels en sa possession, contre 42 pour le joueur.

Dans le pack de textures ASCII, il est représenté par le caractère `M`.

Dans le pack de textures écureuil, il est représenté par l'émoji .

10.3.3 Trompette

Son nom est fixé à *trumpet*. Une trompette est un familier, c'est à dire que c'est une entité attaquante qui suit globalement le joueurs et attaque les monstres qui se rapprochent trop du joueur.

Elle a 20 points de vie et une attaque de 3.

Dans le pack de textures ASCII, elle est représentée par le caractère `/`.

Dans le pack de textures écureuil, elle est représentée par l'émoji .

10.4 Objets

Un objet est une entité présente sur la carte que le joueur peut ramasser. Il lui suffit pour cela de s'approcher, et une fois sur la case de l'objet, celui-ci est placé dans l'inventaire.

Un objet dispose de deux paramètres :

- `held: bool`
  Indique si l'objet est placé dans l'inventaire ou s'il est au sol.
- `held_by: Optional[Player]`
  Si l'objet est dans l'inventaire, renvoie son propriétaire.

Il y a plusieurs types d'objets :

10.4.1 Bombe

Une bombe est un objet que l'on peut ramasser. Une fois ramassée, elle est placée dans l'inventaire. Le joueur peut ensuite utiliser la bombe, via l'inventaire ou après l'avoir équipée, qui fera alors 5 dégâts à toutes les entités attaquantes situées à moins de trois cases au bout de 4 ticks de jeu.

Elle est représentée dans le pack de textures ASCII par le caractère `ç` et dans le pack de textures écureuil par l'émoji . Lors de l'explosion, la bombe est remplacée par un symbole `%` ou l'émoji selon le pack de textures utilisé.

La bombe coûte 4 Hazels auprès des marchands.

10.4.2 Cœur

Un cœur est un objet que l'on ne peut pas ramasser. Dès que le joueur s'en approche ou qu'il l'achète auprès d'un marchand, il récupère automatiquement 5 points de vie, et le cœur disparaît.

Il est représenté dans le pack de textures ASCII par le caractère et dans le pack de textures écureuil par l'émoji .

Le cœur coûte 3 Hazels auprès des marchands.

10.4.3 Potion d'arrachage de corps

Cette potion permet, une fois utilisée, d'échanger toutes ses caractéristiques avec une autre entité aléatoire sur la carte. Cela inclut la force, la position, l'icône, ...

Elle est représentée par les caractères `I` et

Cette potion coûte 14 Hazels auprès des marchands.

10.4.4 Règle

La règle est une arme que l'on peut trouver uniquement par achat auprès d'un marchand pour le coût de 2 Hazels ou dans un coffre. Une fois équipée, la règle ajoute 1 de force à son porteur.

Elle est représentée par les caractères `\` et .

10.4.5 Épée

L'épée est une arme que l'on peut trouver uniquement par achat auprès d'un marchand pour le coût de 20 Hazels ou dans un coffre. Une fois équipée, l'épée ajoute 3 de force à son porteur.

Elle est représentée par les caractères `†` et .

10.4.6 Bouclier

Le bouclier est un type d'armure que l'on peut trouver uniquement par achat auprès d'un marchand pour le coût de 16 Hazels ou dans un coffre. Il s'équippe dans la main secondaire. Une fois équipé, le bouclier ajoute 2 de constitution à son porteur, lui permettant de parer mieux les coups.

Il est représenté par les caractères `D` et .

10.4.7 Casque

Le casque est un type d'armure que l'on peut trouver uniquement par achat auprès d'un marchand pour le coût de 18 Hazels ou dans un coffre. Il s'équippe sur la tête. Une fois équipé, le casque ajoute 2 de constitution à son porteur, lui permettant de prendre moins de dégâts.

Il est représenté par les caractères `0` et .

10.4.8 Plastron

Le plastron est un type d'armure que l'on peut trouver uniquement par achat auprès d'un marchand pour le coût de 30 Hazels ou dans un coffre. Il s'équippe sur le corps. Une fois équipé, le casque ajoute 4 de constitution à son porteur, lui permettant de prendre moins de dégâts.

Il est représenté par les caractères `(` et .

10.4.9 Anneau

Un anneau est un objet que l'on peut trouver uniquement par achat auprès d'un marchand ou dans un coffre. Il s'équippe sur la main secondaire. Une fois équipé, l'anneau ajoute un bonus à une ou plusieurs statistiques du joueur, améliorant sa capacité à se débarasser des monstres.

Il y a plusieurs types d'anneaux :
- **Anneau de coup critique**, qui augmente la chance de coup critique de 20%. Il coute 15 Hazels.
- **Anneau de gain d'expérience amélioré**, qui multiplie le gain d'expérience du joueur par 2. Il coûte 25 Hazels.

Un anneau est représenté par les caractères `o` et .

10.4.10 Monocle

L'anneau est un objet que l'on peut trouver uniquement par achat auprès d'un marchand pour le prix de 10 Hazels. On peut le trouver sinon dans les coffres. Il s'équippe sur la main secondaire.

Une fois porté, il permet de voir les caractéristiques des entités voisines (nom, force, chance de critique, . . .).

Un monocle est représenté par les caractères `ô` et .

10.4.11 Parchemin

Un parchemin est un objet consommable qui se trouve chez un marchand ou dans un coffre. Lorsqu'il est utilisé, il a un effet sur les statistiques du joueur ou des autres entités combattantes. L'intensité de l'effet du parchemin dépend de l'intelligence du joueur.

Il y a plusieurs types de parchemins :
- **Parchemin de dégâts**, qui inflige des dégâts à toutes les entités combattantes qui sont à distance moins de 5 du joueur (ça touche aussi les familiers, mais pas le joueur). Le nombre de points de dégâts est directement l'intelligence du joueur. Il coute 18 Hazels.
- **Parchemin de faiblesse**, qui réduit la force de toutes les entités sauf le joueur de min(1, intelligence//2) pendant 3 tours du jeu. Il coûte 13 Hazels.

Un parchemin est représenté par les caractères `]` et .

10.4.12 Arc

Un arc est une arme à distance qui s'équippe dans la main principale. Pour l'utiliser, il faut appuyer sur la touche de lancer (l de base) puis une touche de direction. Une flèche est alors tirée dans cette direction, et touche le premier ennemi qu'elle rencontre, s'il existe, sur les 3 premières cases dans cette direction.

La flèche fait 4 + dextérité du joueur dégâts. L'arc coûte 22 Hazels chez un marchand. On peut le trouver sinon dans les coffres.

Il est représenté par les caractères `)` et .

10.4.13 Baton de boule de feu

Un baton est une arme à distance qui s'équippe dans la main principale. Pour l'utiliser, il faut appuyer sur la touche de lancer (l de base) puis une touche de direction. Une boule de feu est alors tirée dans cette direction, et touche le premier ennemi qu'elle rencontre, s'il existe, sur les 4 premières cases dans cette direction. Lorsqu'un ennemi est touché, une explosion est affichée sur sa case.

La boule de feu fait 6 + intelligence du joueur dégâts. Le baton coûte 36 Hazels chez un marchand. On peut le trouver sinon dans les coffres.

Il est représenté par les caractères `:` et .

10.5 Entité

Une entité est un élément placé sur la carte. Ce peut être le joueur, un monstre ou bien un objet sur la carte. Chaque entité dispose des attributs suivants :

- `name: str`
  Il s'agit du type de l'entité.
- `y: int`
- `x: int`
  Cela représente les coordonnées de l'entité sur la carte.
- `map: Map`
  Il s'agit de la carte sur laquelle est placée l'entité.

Il existe à l'heure actuelle deux types d'entité : une *entité attaquante* ou bien un objet.

10.6 Entité attaquante

Une entité attaquante (`FightingEntity`) est un type d'entités représentant les personnages présents sur la carte, pouvant alors se battre. Ce peut être un monstre, une entité pacifique ou bien le joueur.

Elles disposent toutes, en plus des paramètres d'entité, des attributs suivants :

- `maxhealth: int`
  Représente la vie maximale de l'entité, qui est aussi la vie de départ.
- `health: int`
  Représente la vie actuelle de l'entité.
- `strength: int`
  Représente la force de l'entité, le nombre de dégâts à faire à chaque coup.
- `intelligence: int`
- `charisma: int`
- `dexterity: int`

— `constitution: int`
Tous ces paramètres sont des statistiques de l'entité, n'ayant pas de réelle influence pour le moment.
— `level: int`
Niveau de l'entité.

Chaque type d'entité disposera de ses propres attributs de départ.

On considère une entité comme morte à partir du moment où sa vie descend en-dessous de 0 point de vie. À ce moment-là, l'entité est retirée de la carte.

Lorsqu'une entité en frappe une autre, celle-ci inflige autant de dégâts qu'elle n'a de force, et autant de points de vie sont perdus.

10.7 Entité pacifique

Une entité pacifique (`FriendlyEntity`) est un cas particulier d'entité attaquante. Contrairement aux montres, elles ne peuvent pas attaquer le joueur.

On peut parler à une entité pacifique en appuyant sur la touche `T` puis en appuyant sur la direction dans laquelle on veut parler à l'entité.

CHAPITRE 11

Pack de textures

Chaque entité et chaque tuile de la carte est associé à un caractère pour être affiché dans le terminal. Cependant, afin de pouvoir proposer plusieurs expériences graphiques (notamment en fonction du support des émojis), différents packs de textures sont proposés.

Il est possible de changer de pack dans les paramètres.

Les packs de textures peuvent influencer la taille que prennent les tuiles, en raison du fait que les émojis ne sont pas monospace.

Les packs de textures sont au nombre de deux :

11.1 Pack ASCII

Chaque tuile fait un caractère de large.

- **Tuiles**
  - Vide : *espace*
  - Mur : `#`
  - Sol : `.`
- **Entités**
  - Joueur : `@`
  - Hérisson : `*`
  - Lapin : `Y`
  - Tigre : `n`
  - Nounours : `8`
  - Tournesol : `I`
  - Marchand : `M`
  - Cœur :
  - Bombe : `o`
  - Explosion : `%`
  - Potion d'arrachage de corps : `S`
  - Épée : `†`
  - Bouclier : `D`

    - Hazel : ¤
    - Plastron : (
    - Pygargue : µ
    - Casque : 0
    - Anneau : ○
    - Trompette : /

11.2 Pack Écureuil

Chaque tuile fait 2 caractères de large pour afficher les émojis proprement.

- **Tuiles**
    - Vide : *espace*
    - Mur :
    - Sol :
- **Entités**
    - Joueur :
    - Hérisson :
    - Lapin :
    - Tigre :
    - Nounours :
    - Tournesol :
    - Marchand :
    - Cœur :
    - Bombe :
    - Explosion :
    - Potion d'arrachage de corps :
    - Épée :
    - Bouclier :
    - Hazel :
    - Plastron :
    - Pygargue :
    - Casque :
    - Anneau :
    - Trompette :

CHAPITRE 12

Paramètres

Il est possible de changer les touches utilisées dans le jeu dans le menu des paramètres.

On peut aussi changer le pack de textures utilisé.

12.1 Touches

Les touches utilisées de base sont :

- **Aller vers le haut** : z
- **Aller vers le haut (secondaire)** : ↑
- **Aller vers le bas** : s
- **Aller vers le bas (secondaire)** : ↓
- **Aller à droite** : d
- **Aller à droite (secondaire)** : →
- **Aller à gauche** : q
- **Aller à gauche (secondaire)** : ←
- **Valider le choix** : Entrée
- **Inventaire** : i
- **Utiliser un objet** : u
- **Équiper un objet** : e
- **Lacher un objet** : r
- **Parler** : t
- **Attendre** : w
- **Utiliser une arme à distance** : l
- **Dancer** : y
- **Utiliser une échelle** : <

12.2 Autres

— **Texture pack utilisé** : parmi ascii et squirrel
— **Langue utilisée** : parmi anglais, français, espagnol, allemand

CHAPITRE 13

Résolution d'erreurs

13.1 Émojis

Le jeu s'exécutant en terminal, il est courant d'obtenir des problèmes d'affichage. Sous Windows, les émojis s'affichent normalement correctement. Il suffit en général d'installer les bons paquets de police.

13.1.1 Sous Arch Linux

Il est recommandé d'utiliser le terminal *xfce4-terminal*. Il suffit d'installer le paquets de polices :

```
sudo pacman -Sy noto-fonts-emoji
```

Le jeu doit ensuite se lancer normalement sans action supplémentaire.

13.1.2 Sous Ubuntu/Debian

À nouveau, le terminal *xfce4-terminal* est recommandé. Le paquet *fonts-noto-color-emoji*.

Toutefois, un problème reste avec l'écureuil. Sous Ubuntu et Debian, le caractère écureuil existe déjà, mais ne s'affiche pas proprement. On peut appliquer un patch qui permet d'afficher les émojis correctement dans son terminal. Pour cela, il suffit de faire :

```
ln -s $PWD/debian/75-fix-squirrel-emojis.conf /etc/fonts/conf.avail/75-fix-squirrel-
↪emojis.conf
ln -s /etc/fonts/conf.avail/75-fix-squirrel-emojis.conf /etc/fonts/conf.d/75-fix-
↪squirrel-emojis.conf
```

Après redémarrage du terminal, l'écureuil devrait s'afficher correctement.

Pour supprimer le patch :

```
rm /etc/fonts/conf.d/75-fix-squirrel-emojis.conf
```

À noter que ce patch est inclus dans le paquet Debian.