Projet RL : Real-Time Strategy

N. Baskiotis, S. Lamprier

2019-2020

On s'intéresse dans ce projet à un jeu de stratégie en temps réel (type Warcraft/Starcraft). Le module utilisé est Deep-RTS, un module encore en développement mais stable, qui a peu de documentation.

1 Module `Deep-RTS`

Pour cloner le github, **utiliser obligatoirement** les commandes suivantes :
`git clone https://github.com/UIA-CAIR/DeepRTS.git --recurse-submodules`. Sinon, le module ne marchera pas.

1.1 Description du jeu

Le module propose un RTS simplifié dont voici les principaux composants (on utilisera les termes anglais pour plus de clarté pour l'interfaçage avec le jeu) :

- deux types de ressources sont disponibles : *gold* (or) et *lumber* (bois) ; ces ressources sont disponibles dans certaines cases de la carte de jeu ;
- les autres types de cases sont *grass* (herbe, on peut se déplacer dessus), *wall* (mur, pas possible de s'y déplacer) et *water* (eau, de même, par contre il est possible de tirer par dessus) ;
- le bâtiment de base est le *townhall* (la mairie), qui permet de produire des *peasants* (fermiers).
- un fermier peut être en mode :
  - *harvesting* (récolter) des ressources (gold ou lumber) ; il se déplace jusqu'à une case de ressources, passe un certain temps dessus puis retourne au townhall pour déposer sa récolte ;
  - *combat* contre un ennemi ou une structure ennemi ;
  - *building* (construire) un bâtiment ;
  - *walking* (marcher) ;
- les unités de combat sont les *footmans* (infanterie, combat rapproché) ou les *archers* (archers, combat eloigné, plus fragile) ;
- chaque unité a un coût de production en gold et en lumber ; par ailleurs, chaque unité consomme tant qu'elle est vivante un certain nombre d'unité de *food* (nourriture) ;
- mis à part le townhall, les autres bâtiments disponibles sont :
  - *farm* : une ferme, qui apporte au joueur 4 unités de nourriture ;
  - *barrack* : un baraquement, qui peut produire des unités de type *footman* (infanterie) ou *archer* (archer)

Tous les bâtiments sont constructibles par les fermiers. Toutes les unités de combats sont productibles par les *barracks*.

Grossièrement, les phases de jeu dans un RTS sont :

- Production de fermiers qui vont récolter des ressources, construire des fermes (pour augmenter la capacité en nombre d'unités)
- Construction de baraquements pour pouvoir produire des unités de combat
- Attaque de l'adversaire (en gardant une économie saine pour pouvoir produire des unités)

Le jeu prend fin lorsqu'un des joueurs n'a plus d'unités et ne peut plus rien produire.

Pour information, voici quelques caractéristiques des unités et bâtiments (dans l'ordre : point de vie, armure, dommage lors d'un combat, portée de l'attaque, coût en bois, en or, en nourriture pour une unité, production de nourriture pour un bâtiment) :

Unité/Bâtiment	Point de vie	Armure	Dommage	Portée	Vitesse	Lumber	Gold	Food
Peasant	30	0	2-9 (2)	1	10	0	400	-1
Footman	60	4	2-9 (4)	1	30	0	600	-1
Archer	40	0	3-9 (2)	4	10	50	500	-1
TownHall	1200	0				250	500	+1
Farm	400	20				250	500	+4
Barrack	800	20				450	700	

1.2 Utilisation du module

Un premier objet `Engine.Config()` permet de créer une configuration du jeu : il est possible d'activer ou désactiver certaines unités, de régler un mode *auto-harvesting* qui fait que les fermiers récolteront automatiquement des ressources quand ils sont inactifs, de régler les ressources initiales etc. Ci-dessous un exemple commenté pour chaque option :

```
from DeepRTS import Engine

engine_config = Engine.Config() # Création de la configuration
engine_config.set_archer(True)  # Autoriser les archers
engine_config.set_barracks(True) # Autoriser les baraquement
engine_config.set_farm(True)     # Autoriser les fermes
engine_config.set_footman(True)  # Autoriser l'infanterie
engine_config.set_auto_attack(False) #Attaquer automatiquement si on est attaqué
engine_config.set_food_limit(1000) # Pas plus de 1000 unités
engine_config.set_harvest_forever(False) # Récolter automatiquement
engine_config.set_instant_building(False) #  Temps de latence ou non pour la construction
engine_config.set_pomdp(False) # Pas de brouillard (partie de la carte non visible)
engine_config.set_console_caption_enabled(False) # ne pas afficher des infos dans la console
engine_config.set_start_lumber(500) # Lumber de départ
engine_config.set_start_gold(500) # Or de départ
engine_config.set_instant_town_hall(False) # Temps de latence ou non pour la construction d'un townh
engine_config.set_terminal_signal(True) # Connaître la fin du jeu
```

On propose d'utiliser exactement cette configuration dans la suite du projet. En particulier, les fermiers ne sont pas en mode récolte automatique et la construction de bâtiments et d'unités est instantanée.

On peut également configurer l'interface graphique :

```
from DeepRTS import python

gui_config = python.Config(render=True, #activer la GUI
                           view=True,
                           inputs=True, #interagir avec un joueur humain
                           caption=True,
                           unit_health=True,
                           unit_outline=True,
                           unit_animation=True,
                           audio=False)
```

Lorsque `inputs` est à vrai (attention, l'attribut dans `gui_config` lui s'appelle `input` !), on peut interagir avec le jeu, en sélectionnant une unité avec le clic gauche de la souris. Le clic droit permet de désigner une case de destination pour l'unité (si ennemi, l'unité la combat, si ressource et l'unité est un fermier, le fermier récolte la ressource). Lorsqu'un bâtiment est sélectionné, la touche `1` permet de construire l'unité 1 (fermier pour un townhall, footman pour un baraquement), la touche `2` de construire l'unité 2 (archer pour un baraquement). Si un fermier est sélectionné, la touche `1` permet de construire un townhall, la touche `2` une ferme et la touche `3` un baraquement.

L'objet `python.Config.Map` contient les cartes disponibles. Le code suivant permet d'initialiser le jeu et de lancer une partie :

```
MAP = python.Config.Map.TWENTYONE
game = python.Game(MAP,n_players=2,engine_config=engine_config,gui_config=gui_config)
game.set_max_fps(250) #augmenter le fps lorsqu'on ne veut pas visualiser le jeu
game.set_max_ups(250)
game.reset()
while not game.is_terminal():
    game.update()
```

1.3 Interfaçage

L'objet `game` contient plusieurs méthodes pour interagir avec le jeu.

L'état du jeu L'état peut être obtenu grâce à la méthode `game.get_state()`. Elle renvoit un tenseur de taille `X x Y x 10`, avec `X` et `Y` la taille de la carte. La dernière dimension du tenseur indique les informations suivantes selon l'index :

- 0 : type de cases : 2 : grass, 3 : wall, 4 : lumber, 5 : water, 6 : gold
- 1 : id du joueur à qui appartient l'unité/bâtiment de la case
- 2 : à 1 si c'est un bâtiment
- 3 : à 1 si c'est une unité
- 4 : type de bâtiment/unité : 1 : peasant, 3 : townhall, 4 : barracks, 5 : footman, 6 : farm, 7 : archer
- 5 : pourcentage de vie
- 6 : état de l'unité : 1 : spawn (apparition), 2 : walk, 3 : despawn (disparition), 4 : harvesting, 5 : building,6 : combat, 7 : dead, 8 : idle (ne fait rien)
- 7 : unité de récolte transportée (si c'est un fermier)
- 8 : Attack Score (un score d'attaque pour l'unité)
- 9 : Defense Score (un score de défense pour l'unité)

Attention, tel quel, l'index 1 est dur à utiliser : le 0 code à la fois l'id du joueur 0 et le fait qu'il n'y a rien dans la case. Il peut être bien de considérer par exemple `state[:,:,1]+state[:,:,2]+state[:,:,3]` ce qui permettra d'avoir un 1 pour le joueur 0 et un 2 pour le jouer 1 (et 0 si pas d'unité/bâtiment dans la case).

Transmettre une action La liste des joueurs est contenue dans `game.players`. La commande `p = game.players[0]` permet de récupérer donc le premier joueur. Il y a deux facons de transmettre une action au jeu. La première est par l'intermédiaire de la méthode `p.do_action(idAction)`. L'action est alors un entier entre 1 et 16 compris, dont voici la liste :

- 1 : PreviousUnit (sélectionner l'unité précédente)
- 2 : NextUnit (sélectionner l'unité suivante)
- 3 : MoveLeft
- 4 : MoveRight
- 5 : MoveUp
- 6 : MoveDown
- 7 : MoveUpLeft
- 8 : MoveUpRight
- 9 : MoveDownLeft
- 10 : MoveDownRight
- 11 : Attack
- 12 : Harvest
- 13 : Build0 (construction/production de la touche 1)
- 14 : Build1 (construction/production de la touche 2)
- 15 : Build2 (construction/production de la touche 3)
- 16 : NoAction

Les commandes 1 et 2 permettent de changer d'unité, les commandes 3 à 15 s'applique à l'unité sélectionnée. Remarque : attaquer ou récolter n'est suivit d'effet que si c'est possible (si l'unité est à proximité d'une unité ennemie ou d'une case de ressources) ; de même, la construction de bâtiment n'est possible que si il y a suffisamment de ressources et de place là où se trouve l'unité.

La deuxième façon de transmettre une action au jeu est par la méthode `p.do_manual_action(idAction,x,y` qui permet de mimer l'action d'un joueur humain avec la souris. Dans ce cas, il n'y a que 3 actions possibles :

— 0 : NoAction
— 1 : Clic gauche de la souris sur la case (x, y)
— 2 : Clic droit de la souris sur la case (x, y)

L'objet `Player` contient par ailleurs les champs suivants :

— `food` et `food_consumption` qui permettent de connaître la nourriture disponible et la nourriture utilisée
— `gold` et `lumber` qui permettent de connaître la quantité de ressources disponibles pour le joueur
— `num_archer`, `num_barrack`, `num_farm`, `num_footman`, `num_peasant`, `num_town_hall` qui permettent de connaître la quantité de chaque unité
— `statistic_damage_done`, `statistic_damage_taken`, `statistic_gathered_gold`, `statistic_gathered_lumber`, `statistic_units_created` qui permettent de savoir depuis le début du jeu le nombre de dommages infligés et reçus, le nombre de ressources collectées et le nombre d'unités créées
— `get_targeted_unit()` qui permet de savoir quelle est l'unité sélectionnée actuellement et `set_targeted_unit_id(id)` qui permet de la définir.

L'objet `Game` contient les champs suivants :

— `get_height()` et `get_width()` qui permettent de connaître la hauteur et la largeur de la carte courante
— `units` qui permet d'avoir la liste d'unités présentes dans le jeu
— `tilemap` qui permet d'avoir la liste des tuiles du jeu (peu utile, toutes les infos sont déjà dans `get_state()`)

Enfin, l'objet `Unit` contient les champs suivants :

— `type` : renvoie le type d'unité
— `get_player()` : renvoie le joueur à qui appartient l'unité
— `gold_carry`, `lumber_carry` qui renvoient la quantité de ressources que l'unité transporte (si c'est un fermier)
— `health` qui renvoie le nombre de points de vie de l'unité.

D'autres méthodes et attributs sont disponibles dans ces objets mais moins utiles. Vous pouvez retrouver l'ensemble des constantes définies dans le jeu dans le fichier `src/Constants.h` (et plein d'autres informations sur le fonctionnement dans ce répertoire contenant le code source C++ du jeu, par exemple les listes de caractéristiques des unités dans `src/unit/UnitManager.cpp`).

2 Travail demandé

Il vous est demandé de définir une ou plusieurs tâches sur le jeu et de les résoudre à l'aide des algorithmes de RL vu ce semestre. Quelques exemples simples de tâches : récolter le plus vite possible dans un temps imparti des ressources, fabriquer le plus rapidement possible un certain nombre d'unités militaires, arriver à une population donnée dans un certain temps limite, etc. C'est également à vous de définir la fonction de récompense en fonction de la tâche. Il sera certainement utile de faire des fonctions auxiliaires et des actions plus haut niveau que ce que le jeu fourni de base. Vous devrez également réfléchir selon la tâche abordée si vous faîtes une IA globale ou une IA pour chaque unité. Vous serez juger sur la pertinence des solutions explorées et l'adéquation des algorithmes utilisées en fonction des tâches définies. Vous rendrez un rapport détaillant les tâches abordées, l'architecture de votre algorithme et les expériences effectuées.

Vous pouvez bien sûr lancer un jeu à un joueur uniquement selon la tâche abordée. Respecter par contre la configuration donnée en section 1.2 sauf si vous définissez une tâche qui requiert sa modification.

Pensez à désactiver l'affichage et à augmenter le fps (nombre d'affichage par seconde) et le ups (nombre d'update par seconde) afin d'accélerer le jeu (avec `game.set_max_fps(fps)` et `game.set_max_ups(ups)`).